CSC 548: Lecture 7

Overview

Imperative features

Liveness

SSA Form

Functional languages

Functional SSA form

Compiling nested methods

Imperative features

What features make a language imperative?

What data features (i.e. on variables)? What control features?

Imperative features

Extends Hobbes with imperative features, e.g.:

```
thread Main {
  let mutable x : Integer = 0;
  label foo: while (true) {
    let mutable y : Integer = 0;
    x := x+1;
    while (true) {
      Out.println ("x = " + $x + ", y = " + $y);
      y := y+1;
      if (y > x) {
        if (x > 5) {
          break foo;
        } else {
          continue foo;
        }
      }
    }
  }
}
```

Imperative features

In the old days... we just did code generation straight off the imperative code.

Slightly better: add var update and jumps to the IL.

Another tack altogether: remove the imperative features! Why?

Liveness

Imperative features make a difference to calculating liveness of variables:

```
let mutable x = 0;
let mutable y = x+1;
Out.println ($y);
x := y+1;
Out.println ($x);
```

Liveness

```
let mutable x = 0;
while (x < 10) {
  x := x+1;
  // Is x live here?
}
x := 0;
```

Liveness

```
thread Main {
  let mutable x = 0;
  let mutable y = 0;
  label foo: while (true) {
    y := 0;
    x := x+1;
    while (true) {
      Out.println ("x = " + $x + ", y = " + $y);
      y := y+1;
      if (y > x) {
        if (x > 5) {
          break foo;
        } else {
          continue foo;
        }
      }
    }
  }
}
```

Liveness

We can't calculate liveness with a one-pass algorithm any more!

How do we calculate liveness of a loop?

```
label loop: while (E) { B1 } B2
```

What is the liveness of `break loop;`?

What is the liveness of `continue loop;`?

SSA Form

What is SSA form?

What is a phi-function? Why do we need phi-functions?

How can we convert a program into SSA form?

Functional languages

What makes a language functional?

What is a pure functional language? An impure language? Is Hobbes pure or impure?

What are the advantages / disadvantages of functional languages over imperative languages?

Functional languages

Extend Hobbes with nested functions:

```
class Fact implements Object {
  method fact (n:Integer) : Integer {
    method loop (result:Integer, i1:Integer) : Integer {
      if (i1 < n) {
        let i2:Integer = i1 + 1;
        let result = result * i2;
        return inner.loop (result, i2);
      } else {
        return result;
      }
    }
    return inner.loop (1, 1);
  }
}
```

Functional SSA form

What is functional SSA form?

Features of functional SSA:

- No phi-functions!
- No imperative features
- Allows inner methods (aka nested functions)

Functional SSA form

To convert to functional SSA form...

1. Remove imperative control flow, replace it with nested methods...

label l: while (E) { B1 } B2 becomes what?

label l: if (E) { B1 } else { B2 } B3 becomes what?

break l; becomes what?

continue l; becomes what?

Functional SSA form

To convert to functional SSA form...

2. Remove imperative data flow, replace it with extra parameters on nested functions...

`let mutable x = e;` becomes what?

`x := e` becomes what?

What about:

```
let mutable x = 0;
method foo () : Unit {
  Out.println ($x);
  x := x+1;
  if (x < 10) { return foo (); } else { return Nothing; }
}
foo ();
```

Functional SSA form

To convert to functional SSA form...

2. Remove imperative data flow, replace it with extra parameters on nested functions...

In general, what about:

```
method bar (Params) : Result {
  B1
}
... bar (Args); ...
```

Functional SSA form

To convert to functional SSA form...

3. Alpha-convert (rename all variables to be unique).

Functional SSA form

An example:

```
method fact (n:Integer) : Integer {
  let mutable i:Integer = 1;
  let mutable result:Integer = 1;
  label loop: while (i < n) {
    i := i + 1;
    result := result * i;
    continue loop;
  }
  return result;
}
```

Functional SSA form

So that was...

1. Replace imperative control flow with nested methods.

2. Replace imperative data flow with extra parameters to methods.

3. Alpha convert.

Note that (2) is expensive - why? What can we do about that expense?

Homework

Implement the transformation to functional SSA form! (Part 2.)

Summary

Programming languages often have imperative features.

Optimizing imperative features can tricky!

One solution is to remove all the imperative features, and do the optimization on an impure functional language: this is Functional SSA.

For code generation for Functional SSA we need to deal with nested methods: either by static links, closure conversion, lambda-lifting, or restricting inner calls to be tail calls.

Next week: Optimizations on Functional SSA