CSE 131 Midterm (Fa16)

Ranjit Jhala

October 28th, 2016

NAME ___________________________________

SID ___________________________________

The exam is **multiple choice**, for each question **circle all valid choices**.

- Each question is worth **5 points**
- You will receive fractional credit for each *correct* choice
- e.g. 1/2 of the points per correct choice, if *two* valid choices.
- You will **lose one point** for each *incorrect* choice.

Problem	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Q1	1	2	3	4	5	6	7
Q2	1	2	3	4	5	6	7
Q3	1	2	3	4	5	6	7
Q4	1	2	3	4	5	6	7
Q5	1	2	3	4	5	6	7
Q6	1	2	3	4	5	6	7
Q7	1	2	3	4	5	6	7
Q8	1	2	3	4	5	6	7
Q9	1	2	3	4	5	6	7

Problem A: Case Study

Suppose we added a `case-of` expression to `cobra`, with the following syntax

```
case e of
  e1      : e1'
  e2      : e2'
  ...
  en      : en'
  default : e'
```

For example, the below should evaluate to

- `100` if `a` equals to `x`,
- `200` if `a` is not equal to `x` but equal to `y`,
- `1000` otherwise.

```
case a of
  x       : 100
  y       : 200
  default : 1000
```

Q1: Representation

To represent `case-of` expressions, we can extend our `Expr` type:

```
type Id = String

data Expr
  = ...
  | Number Integer
  | Var    Id
  | Case   Expr [(Expr, Expr)] Expr
```

What is the Haskell representation of the above example?

1.
   ```
   Case "a"
     [ ("x", Number 100)
     , ("y", Number 200)
     ]
     (Number 1000)
   ```

```
2. Case (Var "a")
     [ (Var "x", Number 100)
     , (Var "y", Number 200)
     ]
     (Number 1000)
```

```
3. Case (Var "a")
     [ (Var "x",       Number 100)
     , (Var "y",       Number 200)
     , (Var "default", Number 1000)
     ]
```

```
4. Case "a"
     [ ("x"      , Number 100)
     , ("y"      , Number 200)
     , ("default", Number 1000)
     ]
```

```
5. Case (Var "a")
     [ (Number 100,  Var "x")
     , (Number 200,  Var "y")
     ]
     (Number 1000)
```

Q2: Immediate Expressions

Suppose you have generic `case-of` expression:

```
case e of
  e1: e1'
  ...
  en: en'
  default: e'
```

Which sub-expressions of the above **must be immediate** for the above to be in A-Normal Form. That is, which sub-expressions must be immediate so that we can generate assembly for `case-of` expressions?

	Imm
1.	`e`
2.	`e1`
3.	`e1'`
4.	`en`
5.	`en'`
6.	`e'`

Compilation

Recall again, the example `case-of` expression from above

```
case a of
  x       : 100
  y       : 200
  default : 1000
```

Assuming that

- `a` is at `[ebp - 4]`
- `x` is at `[ebp - 8]`
- `y` is at `[ebp - 12]`
- `z` is at `[ebp - 16]`

Fill in the blanks so that the following assembly implements the `case-of`:

```
mov eax, [ebp - 4]
label_1:
  cmp eax, [ebp - 8]
  ?1
  mov eax, 100
  ?2
label_2:
  cmp eax, [ebp - 12]
  ?3
  mov eax, 200
  ?4
label_3:
  mov eax, 1000
label_done:
```

HINT: The next questions are all sub-parts of the above. `nop` is the assembly for "do nothing, move to next instruction". Just figure out what the right assembly *should* be, and then mark the right choices. In each case below **there is exactly one right answer**.

Q3, 4: Instructions `?1` and `?2`

	`?1`		`?2`
1.	`nop`	1.	`nop`
2.	`jmp label_2`	2.	`jmp label_2`

	`?1`		`?2`
3.	`jmp label_done`	3.	`jmp label_done`
4.	`je  label_2`	4.	`je label_2`
5.	`je  label_done`	5.	`je label_done`
6.	`jne label_2`	6.	`jne label_2`
7.	`jne label_done`	7.	`jne label_done`

Q5, 6: Instruction `?3` and `?4`

	`?3`		`?4`
1.	`nop`	1.	`nop`
2.	`jmp label_3`	2.	`jmp label_3`
3.	`jmp label_done`	3.	`jmp label_done`
4.	`je  label_3`	4.	`je label_3`
5.	`je  label_done`	5.	`je label_done`
6.	`jne label_3`	6.	`jne label_3`
7.	`jne label_done`	7.	`jne label_done`

Problem B: Stack Allocation

Consider the expression

```
let a =
        let x = 1
        in
        let y = x + 1
        in
        let z = y + 2
        in
          z + 3
in
let b = a + 1
in
    b + 2
```

Q7: Stack Positions

At what **positions** on the stack are the binders (variables) of the above expression stored?

	x	y	z	a	b
1.	1	2	3	4	5
2.	1	2	3	1	2
3.	3	2	1	5	4
4.	3	2	1	4	5
5.	3	2	1	2	1

Q8: How deep is the stack?

How many *slots* do we need to allocate on the stack to compile the above expression? (i.e. what should `countVars` return for the above expression?)

	Slots
1.	1
2.	2
3.	3
4.	4
5.	5

Problem C: Boolean Comparisons

Recall that in `cobra` we represent **booleans** as 32-bit values whose **Most Significant Bit** (MSB) is `1` for `true` and `0` for `false` i.e. the values have the HEX representation:

Value	Representation
`true`	`0x80000001`
`false`	`0x00000001`

Suppose we want to compute the result of the comparison

```
arg1 < arg2
```

In lecture we saw how to do so using the assembly comparisons and jumps.

Q9: Fast comparisons by bit twiddling

Here's a *different* and *simpler* approach, that relies on the observation:

> the MSB of 32-bit value is 1 exactly when the value is negative.

```
mov eax, arg1
sub eax, arg2
?1  eax, ?3
?2  eax, ?4
```

How should we fill in the values of `?1, ?2, ?3, ?4` so that we get a sequence of assembly such that at the end, the value in `eax` is `true` if `arg1 < arg2` and `false` otherwise?

NOTE: Assume there are no overflows when doing the subtraction.

	?1	?2	?3	?4
1.	and	or	0x80000000	0x00000001
2.	or	and	0x80000000	0x00000001
3.	and	or	0x00000001	0x80000000
4.	or	and	0x00000001	0x80000000
5.	and	or	0xFFFFFFFF	0x00000001