

Homework # 11

due 2008/4/29

1 Stack Frames

Tail-call optimization, and tail recursion optimization in particular, are important optimizations for functional programming languages and also make sense for object-oriented languages such as Cool. A *tail call* in Cool is a dispatch or static dispatch that is the last thing done in a method. In other words, it represents a call whose value will be returned. In the past students did this homework very poorly because they didn't think. This year, I'm including a lot of examples and ungraded questions. Skip them at your peril!

Example:

```
fact(n : Integer) : Integer :=
  if n > 0 then n * fact(n) else 1;

test(n : Integer) : Sieve := begin
  if prime * (n div prime) = n then
    self
  else if void == next then
    out_int(n); out_string(" ");
    next := new Sieve(n)
  else
    next.test(n)
  fi
fi
end;
```

In the two methods given above, the following expressions are “tail” expressions, and no others are:

if n > 0 then n * fact(n) else 1	COND
n * fact(n)	MUL
1	INT_CONST
if prime * (n div prime) = n then ... fi	COND
self	OBJECT
if void == next then ... fi	COND
out_int(n); out_string(" "); next := new Sieve(n)	BLOCK
next := new Sieve(n)	ASSIGN
next.test(n)	DISPATCH

Notice that only one tail expression is also a “dispatch.” In other words, there is only *one* tail call anywhere in these examples. (Of course a different method could have multiple tail calls, but only one can be executed.)

Ungraded questions:

- Is the fact that an expression is “tail” or not be determined internally (synthesized) or from the context (inherited)? The fact that 1 is a tail expression but that 0 is not in the above examples should make the answer to this question pretty easy.
- If we have an expression $x + y$, then can either x or y be tail? Think about whether anything needs to be done after they are evaluated and before the method returns.
- If we have an expression `foo(x,y)`, then can either x or y be tail? Do we need to do anything after evaluating y before being done?
- If we have a while loop, `while foo() loop bar() pool`, can either `foo()` or `bar()` be tail? Do we need to do anything after they finish evaluation?

Graded questions:

- (a) Give an attribute grammar for Cool expressions and branches where each expression or branch has an `isTail` attribute true if the method does nothing more after this expression is done. In other words, if this expression represents what the method will return. All expression nodes get this attribute, but a dispatch or static dispatch node so marked will be a “tail call.” For example, if the method body is

```
if x < y then foo(y) else x+1 fi
```

then the `isTail` attribute is true for the `cond` node, the `dispatch` node (a tail call) and the `add` node. It is false for all `object` nodes here and the `lt` and `int_const` nodes. Think: is `isTail` an inherited or synthesized attribute?

- (b) A tail-call optimization combines the effect of returning and calling so that the stack frame of the caller is removed and the frame of the callee sits directly on the caller of the caller’s frame. This is possible because the caller will simply return whatever the callee returns, so we can remove the “middleman.” Our Cool compiler has the caller push arguments onto the stack, but the callee pop them off. This facilitates tail-call optimization. How? (You may need to answer the following question to discover the reason.)

- (c) Currently a static dispatch followed by the epilogue looks like:

```
[push each argument]
[code for receiver]
[code to test whether receiver is void]
jal Class.method
...
lw      $fp FS($sp)
lw      $s0 FS-4($sp)
lw      $ra FS-8($sp)
addiu   $sp $sp FS+4*NP
jr      $ra
```

What would a tail-call look like if we had tail-call optimization? Hint: it is messy. Write out the template that combines the static dispatch call with the epilogue. (A dispatch can be done as a tail call in exactly the same way, but to keep it simple, you just need to handle static dispatches here.)

2 Pipelines

Analyze the code generated for `Object.equals`. How many cycles does the routine require in the case that it returns true (and there are no cache misses)? Show the schedule of pipeline stages for each instruction.

Hand-optimize the entire code for this method (only!). List the resulting code. Now how many cycles does it take? Again, show your work. (Also check that the program still works by running it in Spim!)