NOTES 11: μOCAML COMPILER

Cameron Moy

Thursday July 11th

1 λ syntax and semantics

Example. Reduce $(\lambda x.(\lambda y.x\ y)\ w)\ y$ to normal form.

Proof. We could do this proof as a series of trees, but this is a lot of writing. Typically we just show the series of $\Rightarrow$ directly, underlining the redex.

$$\begin{aligned}(\lambda x.\underline{(\lambda y.x\ y)}\ w)\ y &\Rightarrow^{\alpha} \underline{(\lambda x.(\lambda z.x\ z)\ w)\ y} \\ &\Rightarrow^{\beta} \underline{(\lambda z.y\ z)\ w} \\ &\Rightarrow^{\beta} y\ w\end{aligned}$$

Notice that we are required to α convert in this case, otherwise we are prone to variable capture. ❑

Since $\Rightarrow$ is a relation (and hence non-deterministic), there is actually another way to do this proof.

Proof.

$$\begin{aligned}(\lambda x.\underline{(\lambda y.x\ y)\ w})\ y &\Rightarrow^{\beta} \underline{(\lambda x.x\ w)\ y} \\ &\Rightarrow^{\beta} y\ w\end{aligned}$$

❑

Notice that we get the exact same answer. This is actually a guarantee.

2 Reduction strategies

This is all well-and-good, but non-determinism is bad. If we want to write a λ calculus interpreter we need to define a deterministic strategy. We can describe such a strategy informally.

1. The call-by-value strategy requires that an argument must be evaluated before a β reduction can happen.

2. The call-by-name strategy states that β reductions happen without evaluating the argument at all.

3 It was λ the whole time

Pause for a moment. Doesn't this look familiar? NanOCaml is really just the λ calculus in disguise. A couple differences are worth noting.

1. Our syntax had `fun` instead of λ, an `->` instead of ., and extra parentheses (to make parsing easier). In the actual λ calculus we omit most parentheses and use two conventions to disambiguate: λ bodies extend as far as possible, and application left associates.

2. We choose our semantics last time to implement a CBV reduction strategy. We could've made a different choice and would've gotten a different semantics.