Prop 10.3. If $e_0 \xrightarrow{*} e_1$, $e_0 \xrightarrow{*} e_2$ and e_1 and e_2 are β-normal forms, then $e_1 \equiv e_2$. (i.e., e_1 and e_2 are α-equivalent).

Proof. By Prop 10.2, $\exists\, e_3$ s.t. $e_1 \xrightarrow{*} e_3$ and $e_2 \xrightarrow{*} e_3$.
Since e_1 and e_2 are β-normal forms, $e_1 \equiv e_3$ and $e_2 \equiv e_3$.
$\therefore\ e_1 \equiv e_2$. □

(6) One natural question is whether we can find a good strategy to use the nondeterminism in the third "Contextual Closure" rule, so that if an expression e can be reduced to a normal form, this strategy indeed transforms e to such a normal-form expression.

To see this issue, consider the following two reduction sequences:

$$(\lambda u.\lambda v.v)\,((\lambda x.x\,x)(\lambda x.x\,x)) \to \lambda v.v$$

$$(\lambda u.\lambda v.v)\,((\lambda x.x\,x)(\lambda x.x\,x)) \to (\lambda u.\lambda v.v)\,((\lambda x.x\,x)(\lambda x.x\,x))$$

$$\to \cdots$$

Only the first gives an expression in the normal form.

(7) <u>The normal-order reduction</u> is a particular way of using the "Contextual Closure" rule. It picks a redex (β-redex) in an expression e that is not included in any other redex. Also, if there are multiple such outermost redexes, it picks the <u>leftmost</u> one. In our example, the normal-order reduction doesn't pick the redex $(\lambda x.x\,x)(\lambda x.x\,x)$ because it is included in the redex $(\lambda u\ldots)((\lambda x..)(\lambda x..))$. The normal-order reduction is also called <u>outermost</u> <u>leftmost</u> reduction.

Prop 10.4. If $e \xrightarrow{*} e'$ for some normal-form e', then $e \xrightarrow{*}_{normal} e'$ where $\to_{normal}$ means the contraction relation of the normal-order reduction.

4. Normal-Order Evaluation and Eager Evaluation.

(1) In functional languages, we use restricted versions of the reduction relation to specify how function calls should be handled. We will look at two well-known restrictions used in Haskell and Ocaml, and call them normal-order evaluation and eager evaluation.
Note that we use the word "evaluation" instead of "reduction" or "contraction".

(2) Both normal-order and eager evaluations are defined for closed expressions only, i.e., expressions that do not have any free variables. Also, they are formalised as big-step semantics where the evaluation relation $\Rightarrow$ transforms an expression to a final result in one go, instead of in multiple steps in the reduction relation. Finally, these evaluations do not contract any subexpressions inside lambda. Thus, their results might not be normal forms. They will instead be a <u>canonical form</u>.

(3) Normal-order evaluation $\Rightarrow$:

$$e \Rightarrow z$$

closed expression : expression in the canonical form.
(i.e., lambda expression)

<u>A canonical form z</u> is a lambda expression.