Church's lambda calculus reduces all of computation to what?

The definition and application of functions.

What is the bridge from lambda calculus to a language like ML or Haskell?

The addition of features definable in the lambda calculus itself.

Some are easily defined. E.g., syntax for numbers and

collections.

Others are more challenging. E.g., mutable reference cells.

What are the three kinds of terms in lambda calculus?

Variable: v Abstraction: \$\lambda\$ x.t

Applications: t t

What's the difference between an *internal* and *external* language?

An external language contains derived forms and can be translated to an internal language which uses only core

features.

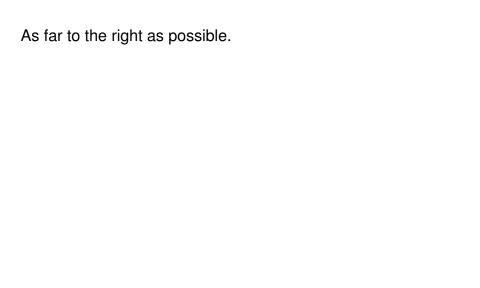
Which way does function application associate?

To the left.

The following two are equivalent:

```
fgu
(fg)u
```

The body of a lambda abstraction extends to where?



What's the difference between a *bound* and *free* variable.

What's a *closed term*?

A variable is free if it is not bound by an enclosing abstraction.

Closed terms (aka *combinators*) have no free variables.

Describe computation in the lambda calculus.

A redex term:

```
(\lambda x.t_1) t_2
```

Evaluation by beta reduction: a new term, namely $$t_1$$ with each instance of x in it replaced by \$t2\$.

What is an evaluation strategy?

Do not confuse with ...

A rule for determining which redex in a term can be evaluated when.

... calling convention which is unrelated, even though some of the evaluation strategies have "call" in their name. Name some evaluation strategies.

Which is the most popular?

- Full beta-reduction

Call-by-value is the most common.

- Normal order
- Call-by-nameCall-by-value

What is *full beta-reduction*?

An evaluation strategy which allows an redex in a term to be

evaluated in the next step.

What is the *normal order* strategy?

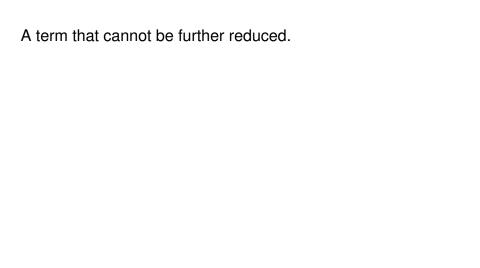
An evaluation strategy in which only the outermost redex can be evaluated in the next step.

What is the *call-by-name* strategy?

An evaluation strategy that follows the normal order strategy but does not allow reductions inside the body of an

abstraction.

What is a *value*?



What is the *call-by-value* strategy?

An evaluation strategy that follows the normal order strategy but in which redexes are reduced by first reducing its

argument to a value.

What's the difference between *strict* and *non-strict* evaluation strategies?

In strict evaluation strategies arguments to abstractions are

evaluated regardless of whether they are used in the body of

a function. In a non-strict (aka *lazy*) evaluation strategy an argument may not be evaluated if it's not used in the body. What evaluation strategy does Haskell use?

Call-by-need, which is call-by-value with memoization. More specifically, after an argument is evaluated all other instances

of that argument will be replaced with the value.