
Midterm Concepts

- ❖ Recursion versus iteration
- ❖ Tail-recursion
- ❖ Higher order functions
- ❖ First class functions
- ❖ Quote and symbols
- ❖ Scope and binding environments
- ❖ Case analysis using match
- ❖ Defining data structures using struct
- ❖ Types
- ❖ Store-passing style

Recursion versus tail-recursion

Tail-recursive: a recursive function in which no computation is done after the return of the recursive call (all the work is inside the recursive call, so there's no need to build a recursive stack).

Higher-order functions

A **higher-order function** is a function that takes another function as an argument.

First-class functions

A language has **first-class functions** if it treats functions the same as other values in the language. Critically, it allows them to be (1) named and (2) to appear in the same syntactic environments as other values (as arguments to functions, as return values, etc).

Quote and symbols

Quote is a way to express data literals. An expression that is quoted is turned into a **symbol** and is not evaluated.

Scope and binding environments

An **environment** is a table of **bindings** that associate variables with values.

A variable is bound in the **scope** of an environment.

Example binding environments: functions, let, letrec, match clauses that name subexpressions...

Case analysis using match

Match is a language construct that makes case analysis easier. **Case analysis** means breaking a procedure into a finite set of cases that define the result based on the conditions.

Types

Types define an ontology: all of the types of a language, taken together, describe what kinds of things are in the language.

A type can be thought of a collection of values for which a particular group of functions is well-defined.

The term type is often reserved for data structures that come predefined in the language; however, user-defined data structures can also be considered types.

Defining data structures

Struct is a Racket language construct for defining new data structures. It specifies the number and names of the data structure's fields, and provides constructor, member, and field-getter functions.

```
(struct my-data-struct (field1 field2))
```

The struct constructor does not validate its input.

Defining data structures

Store-passing style is a way of mimicking mutation in a pure functional programming language. A **store** is a kind of dummy memory; variables are always evaluated alongside a store. By changing which store is passed along with the variable, it appears that the value of the variable itself changes.