
CPSC 110 Concepts Review

CPSC 110

Peyton Seigo

2018-11-5

CPSC 110 Concepts Review

Concepts I need to practice

- Backtracking: if not false
- Binary search trees
- Generative recursion

Useful built-in functions

- `(string-length s) -> Natural`
- `(substring s i j) -> String`
- `(string-ith s i) -> String (one long)`
- `(string-downcase s) -> String`
- `(string-upcase s) -> String`

Concepts

Core recipes:

- HtDF
- HtDD
- HtDW
 - Main function: `(@template htdw-main)`

Data driven templating:

- **Compound data**
- **Ref**
- **Self-ref** (lists)
 - **Naturals**: treating naturals as lists
- **Mutual-ref** (lists of non-primitive types)
- **Helpers**
 - `fn-composition?`
- **Binary Search Trees**
- Cross-product tables: **Two one-of types**
- **Local**
 - 4 uses of local:
 - * Encapsulation
 - * Reduce recomputation
 - * Readability, D.R.Y.

* To pass to an abstract function

- **Abstraction**
- **Generative Recursion**
 - **Search** w/ genrec

(require spd/tags):@ tags

For HtDF:

- (@HtDF FunctionName)
- (@signature Type1 Type2 ... -> ResultType)
- (@template s1 s2 ...)
 - s is a source for a template

Sources for @template:

- TypeName
 - Name of type the template is based on.
 - For encapsulation: Separate TypeName for each encapsulated function.
- add-param
 - Additional parameters are treated as atomic data.
 - Add parameter to each ..., like (... ad-t1 ad-t2) etc.
- htdw-main
 - main fn in HtDW, with a call to big-bang.
- fn-composition
 - Composition of calls to 2+ helper functions.
- backtracking search
- 2-one-of
 - Cross-product table.
 - Possible case reduction.
- encapsulated
 - Encapsulation of 2+ fns.
 - Usually mutually recursive.
- use-abstract-fn
 - Call to 1+ abstract fns, either built in or user defined.
 - If more than 1, use fn-composition
- genrec
 - Generative recursion.
- bin-tree, arb-tree

- Requires use of `genrec`, and indicates that template is a traversal of a generated binary or arbitrary-arity tree.
- `accumulator`
 - 1+ accumulators.
- `for-each`
 - Call to `for-each`.

Template for key & mouse handlers use the *large enumeration rule*. So the tag and template look like this:

```
(@template KeyEvent add-param)
(define (handle-key ws ke)
  (cond [(key=? ke " ") (... ws)]
        [else (... ws)]))
```