
LISTS

Check if x is in list xs

```
if (!is_null(member(x, xs))) { }
```

Remove Duplicates

```
function remove_duplicates(xs) {
    return accumulate(
                (curr, wish) => pair(curr, filter(x => x !== curr, wish)),
                null, xs);
}
```

Permutations

```
function permutations(ys) {
    // list => list of lists
    return is_null(ys)
        ? list(null)
        : accumulate(append, null,
            map(x => map(p => pair(x, p),
                         permutations(remove(x, ys))),
                ys));
}
```

ARRAYS

Reverse Index

```
A[len - i - 1];
```

List-array conversion

```
function list_to_array(xs) {
    const A = [];
    let i = 0;
    while (!is_null(xs)) {
        A[i] = head(xs);
        xs = tail(xs);
        i = i + 1;
    }

    return A;
}

function array_to_list(A) {
    function helper(i) {
        if (A[i] === undefined) {
            return null;
        } else {
            return pair(A[i], helper(i + 1));
        }
    }

    return helper(0);
}
```

"Get Out"

```
function get_out(A, x, y) {
    // removes a subarray from index x to y (inclusive)
    // returns a pair(subarray, removed_array)
    const B = [];
    const A_out = [];

    for (let i = 0; i < array_length(A); i = i + 1) {
        if (i < x) {
            B[i] = A[i];
        } else if (x <= i && i <= y) {
            A_out[i - x] = A[i];
        } else {
            // leaves an undefined element
            B[i - y + x] = A[i];

            // no undefined
            B[i - y + x - 1] = A[i];
        }
    }

    return pair(A_out, B);
}
```

STREAMS

Visualizers

```
function stream_to_list_n(S, n) {
    if (n === 0 || is_null(head(S))) {
        return null;
    } else {
        return pair(head(S), stream_to_list_n(stream_tail(S), n - 1));
    }
}

function stream_to_array_n(S, n) {
    const arr = [];

    for (let i = 0; i < n; i = i + 1) {
        let value = stream_ref(S, i);
        if (value === undefined) {
            break;
        } else {
            arr[i] = value;
        }
    }

    display(arr);
    return arr;
}
```

Map, Filter, Accumulate

```
function array_map(f, A) {
    // destructive and returns
    for (let i = 0; i < array_length(A); i = i + 1) {
        A[i] = f(i);
    }
    return A;
}

function array_filter(f, A) {
    // non-destructive and returns
    let j = 0; const B = [];

    for (let i = 0; i < array_length(A); i = i + 1) {
        if (f(A[i])) {
            B[j] = A[i];
            j = j + 1;
        }
    }

    return B;
}

function array_accumulate(f, initial, A) {
    // non-destructive and returns
    const len = array_length(A);
    for (let i = 0; i < len; i = i + 1) {
        initial = f(A[len - i - 1], initial);
    }

    return initial;
}
```

Append

```
function append_arrays(a1, a2) {
    if (!is_array(a1)) { a1 = [a1]; }
    if (!is_array(a2)) { a2 = [a2]; }

    const l1 = array_length(a1);
    const l2 = array_length(a2);

    const A = [];

    for (let i = 0; i < l1 + l2; i = i + 1) {
        if (i < l1) {
            A[i] = a1[i];
        } else {
            A[i] = a2[i - l1];
        }
    }
    return A;
}
```

Misc

makeup_amount

```
function makeup_amount(x, coins) {
    if (x===0) {
        return list(null);
    } else if (x < 0 || is_null(coins)) {
        return null;
    } else {
        // Combinations that do not use the head coin
        const combi_A = makeup_amount(x, tail(coins));

        // Combinations that do not use the head coin
        // for the remaining amount
        // I do not understand...
        // OHHHH IT WORKS LIKE THIS, yeah it is kinda redundant
        const combi_B = makeup_amount(x - head(coins), tail(coins));

        // Combinations that use the head coin
        const combi_C = map(wish => pair(head(coins), wish), combi_B);

        return append(combi_A, combi_C);
    }
}
```

count pairs

```
function count_pairs(xs) {
    let pairs = null;
    const is_in = (ys, a) => !is_null(member(a, ys));

    function helper(xs) {
        if (!is_pair(xs) || is_in(pairs, xs)) {
            return 0;
        } else {
            pairs = pair(xs, pairs);
            return 1 + helper(head(xs)) + helper(tail(xs));
        }
    }
    return helper(xs);
}
```