Chapter 84: Tail Call Optimization

Section 84.1: What is Tail Call Optimization (TCO)

TCO is only available in strict mode

As always check browser and JavaScript implementations for support of any language features, and as with any JavaScript feature or syntax, it may change in the future.

It provides a way to optimise recursive and deeply nested function calls by eliminating the need to push function state onto the global frame stack, and avoiding having to step down through each calling function by returning directly to the initial calling function.

```
function a(){
   return b(); // 2
}
function b(){
   return 1;  // 3
}
a(); // 1
```

Without TCO the call to `a()` creates a new frame for that function. When that function calls `b()` the `a()`'s frame is pushed onto the frame stack and a new frame is created for function `b()`

When `b()` return to `a()` `a()`'s frame is popped from the frame stack. It immediately return to the global frame and thus does not use any of the states save on the stack.

TCO recognises that the call from `a()` to `b()` is at the tail of function `a()` and thus there is no need to push `a()`'s state onto the frame stack. When `b(0)` returns rather than returning to `a()` it returns directly to the global frame. Further optimising by eliminating the intermediate steps.

TCO allows for recursive functions to have indefinite recursion as the frame stack will not grow with each recursive call. Without TCO recursive function had a limited recursive depth.

> **Note** TCO is a JavaScript engine implementation feature, it cannot be implemented via a transpiler if the browser does not support it. There is no additional syntax in the spec required to implement TCO and thus there is concern that TCO may break the web. Its release into the world is cautious and may require browser/engine specific flags to be set for the perceivable future.

Section 84.2: Recursive loops

Tail Call Optimisation makes it possible to safely implement recursive loops without concern for call stack overflow or the overhead of a growing frame stack.

```
function indexOf(array, predicate, i = 0) {
    if (0 <= i && i < array.length) {
        if (predicate(array[i])) {  return i; }
        return indexOf(array, predicate, i + 1); // the tail call
    }
}
indexOf([1,2,3,4,5,6,7], x => x === 5); // returns index of 5 which is 4
```