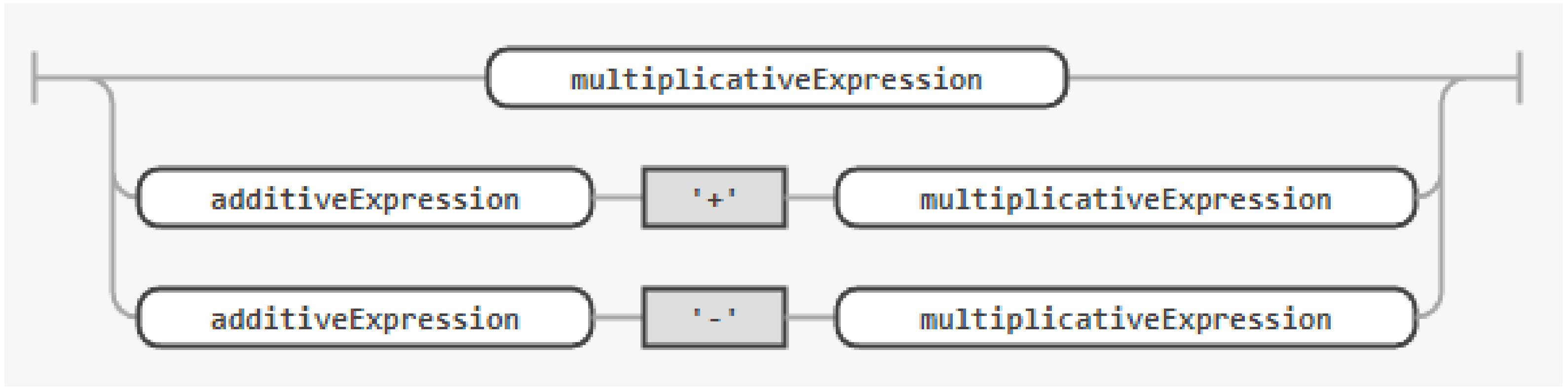
Realisierung eines C-Interpreters mit JavaScript

Patrick Lukas Starzynski

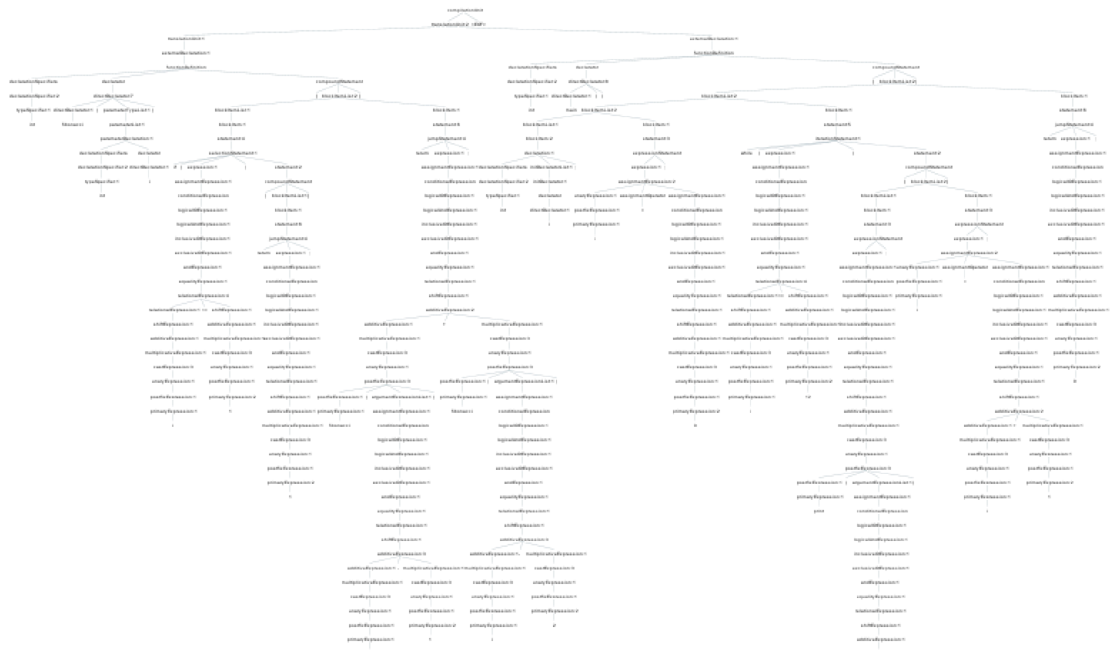
Railroad-Diagramm für Multiplikation



C-Code

- Einfacher Code, riesige Syntaxbäume

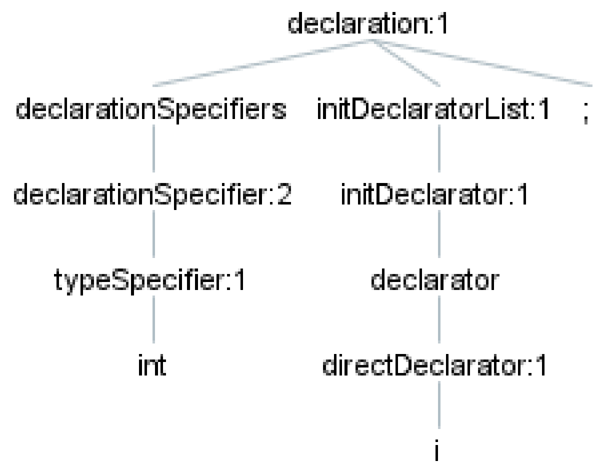
```c
int fibonacci(int i) {
    if (i <= 1) {
        return 1;
    }
    return fibonacci(i-1) + fibonacci(i-2);
}

int main()
{
    int i;
    i = 0;
    while (i <= 12) {
        print(fibonacci(i));
        i = i + 1;
    }
    return 0;
}
```



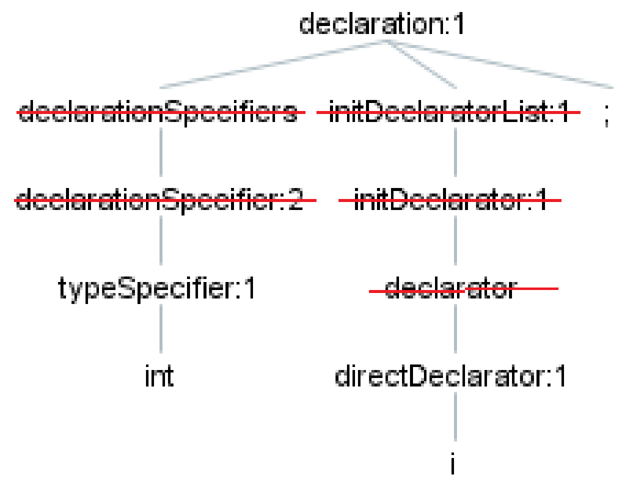
- Schwierig zu interpretiereren

Minimierung der Bäume

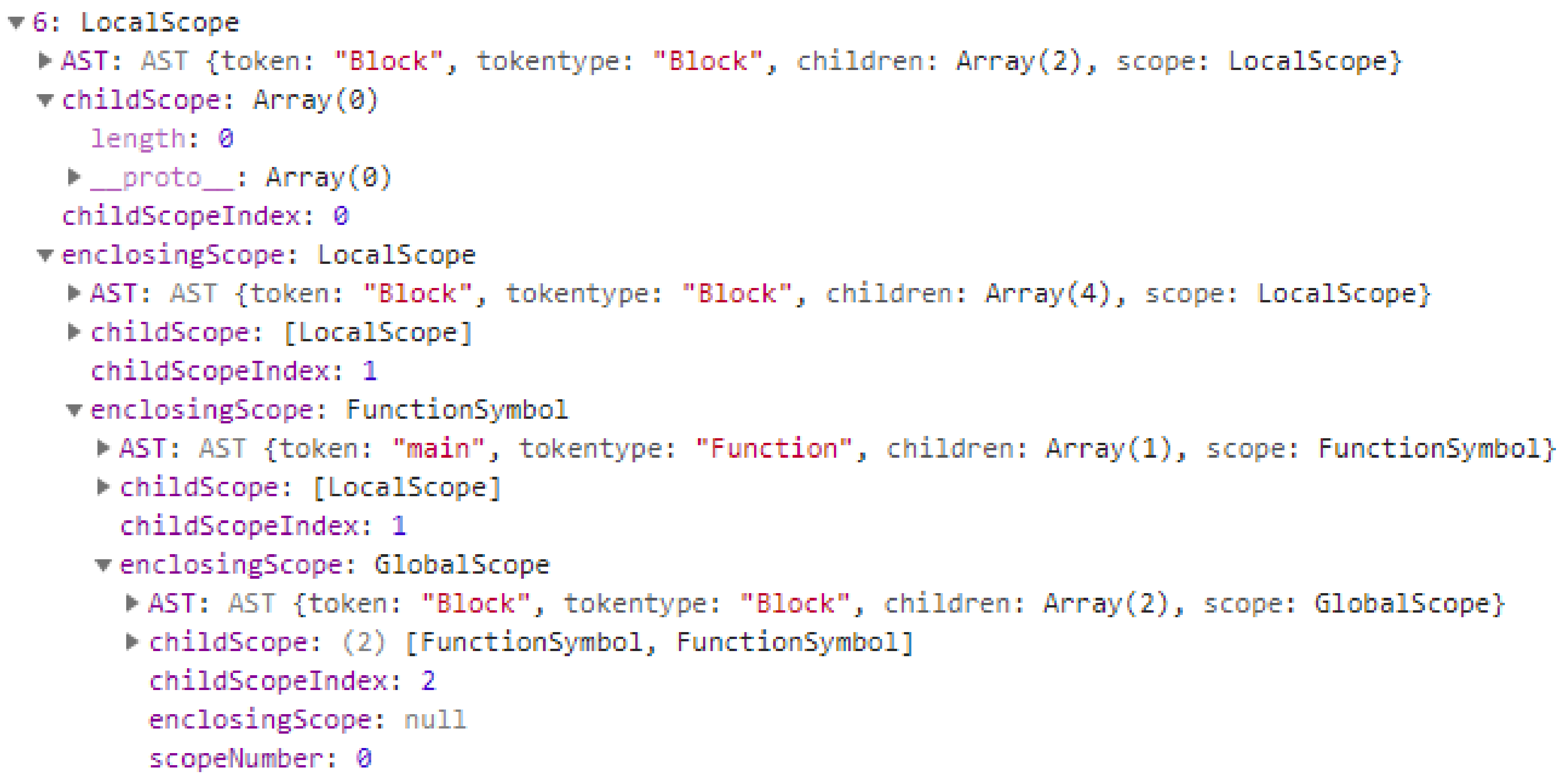
Vorher



Nachher

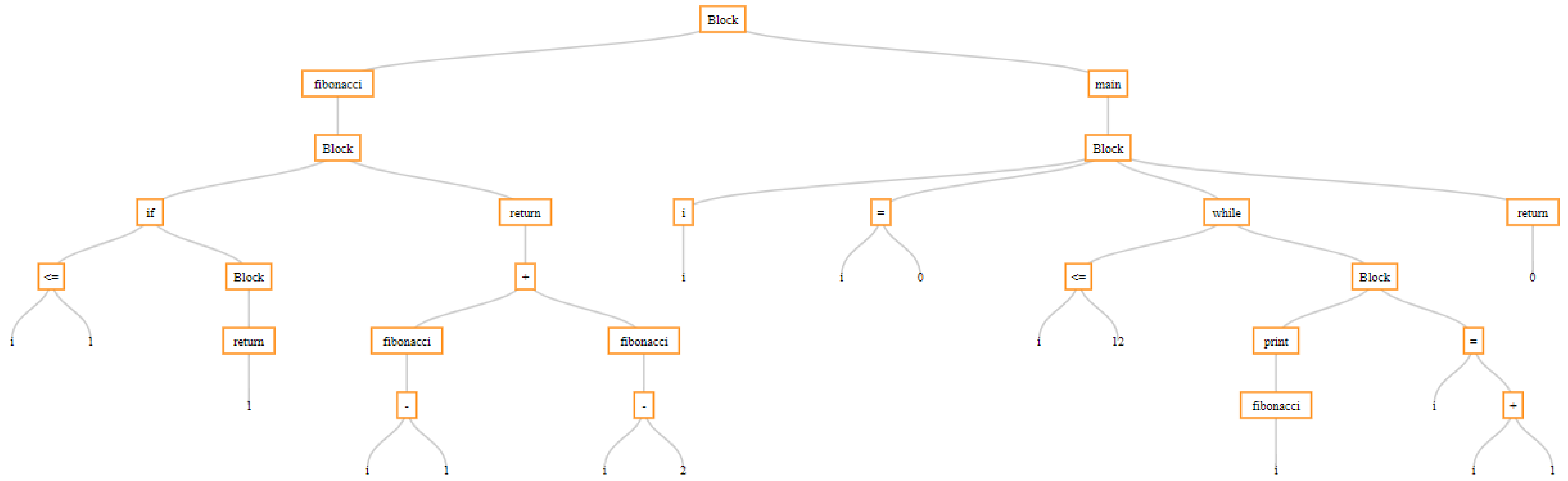


Symboltabelle – If-Statement

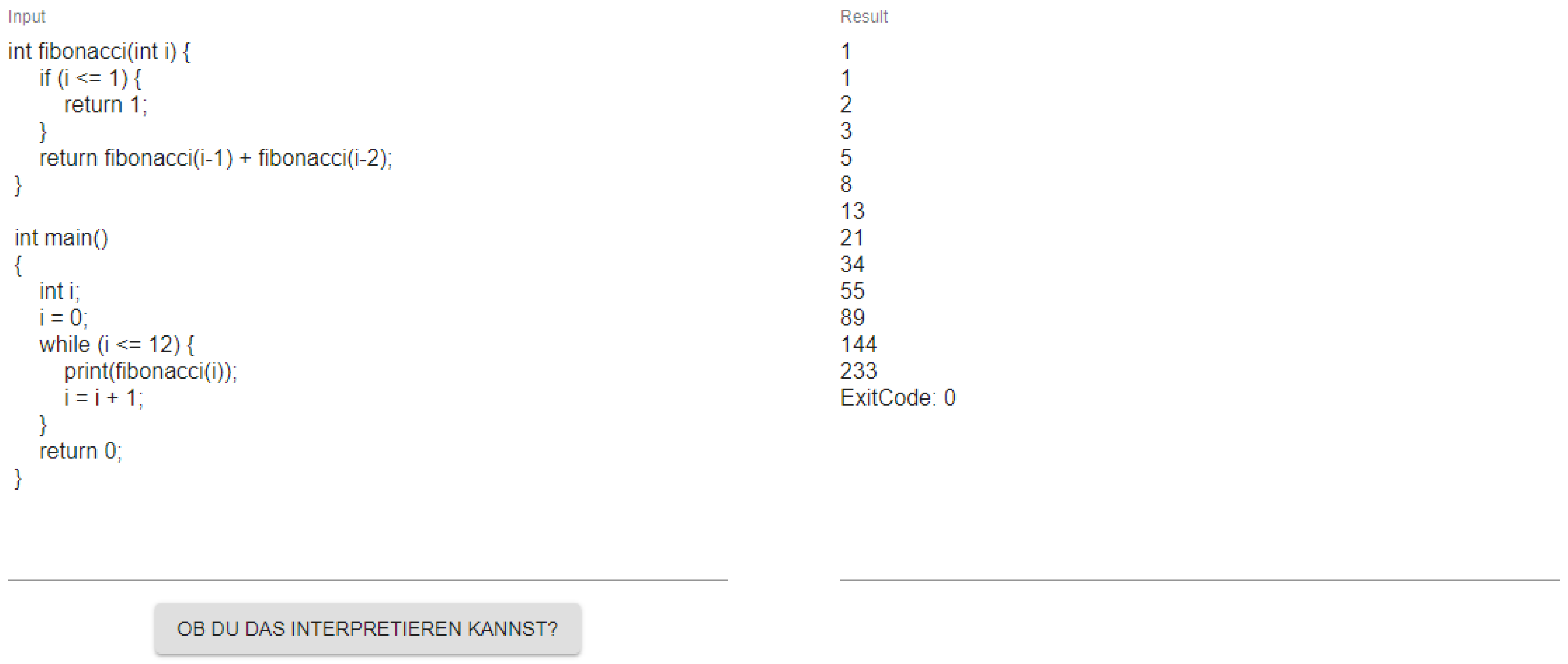


```
▼6: LocalScope
  ▶AST: AST {token: "Block", tokentype: "Block", children: Array(2), scope: LocalScope}
  ▼childScope: Array(0)
     length: 0
    ▶__proto__: Array(0)
   childScopeIndex: 0
  ▼enclosingScope: LocalScope
    ▶AST: AST {token: "Block", tokentype: "Block", children: Array(4), scope: LocalScope}
    ▶childScope: [LocalScope]
     childScopeIndex: 1
    ▼enclosingScope: FunctionSymbol
      ▶AST: AST {token: "main", tokentype: "Function", children: Array(1), scope: FunctionSymbol}
      ▶childScope: [LocalScope]
       childScopeIndex: 1
      ▼enclosingScope: GlobalScope
        ▶AST: AST {token: "Block", tokentype: "Block", children: Array(2), scope: GlobalScope}
        ▶childScope: (2) [FunctionSymbol, FunctionSymbol]
         childScopeIndex: 2
         enclosingScope: null
         scopeNumber: 0
```

AST


Block
fibonacci
main
Block
Block
if
return
i
=
while
return
<=
Block
+
i
i
0
<=
Block
0
i
1
return
fibonacci
fibonacci
i
12
print
=
1
-
-
fibonacci
i
+
i
1
i
2
i
i
1

Interpreter



```
int fibonacci(int i) {
    if (i <= 1) {
        return 1;
    }
    return fibonacci(i-1) + fibonacci(i-2);
}

int main()
{
    int i;
    i = 0;
    while (i <= 12) {
        print(fibonacci(i));
        i = i + 1;
    }
    return 0;
}
```

```
1
1
2
3
5
8
13
21
34
55
89
144
233
ExitCode: 0
```