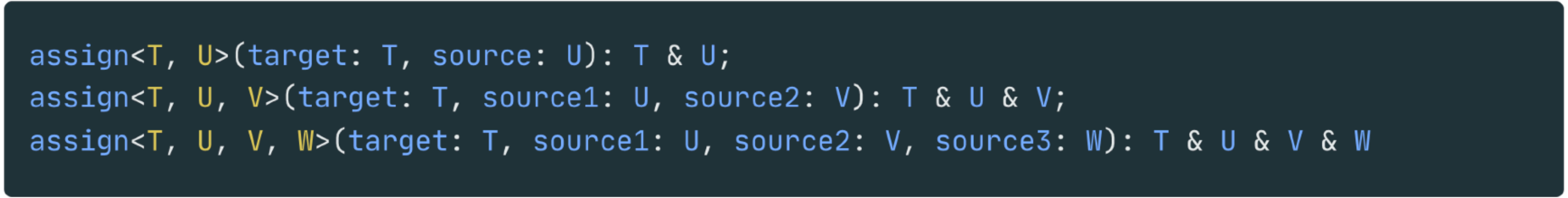
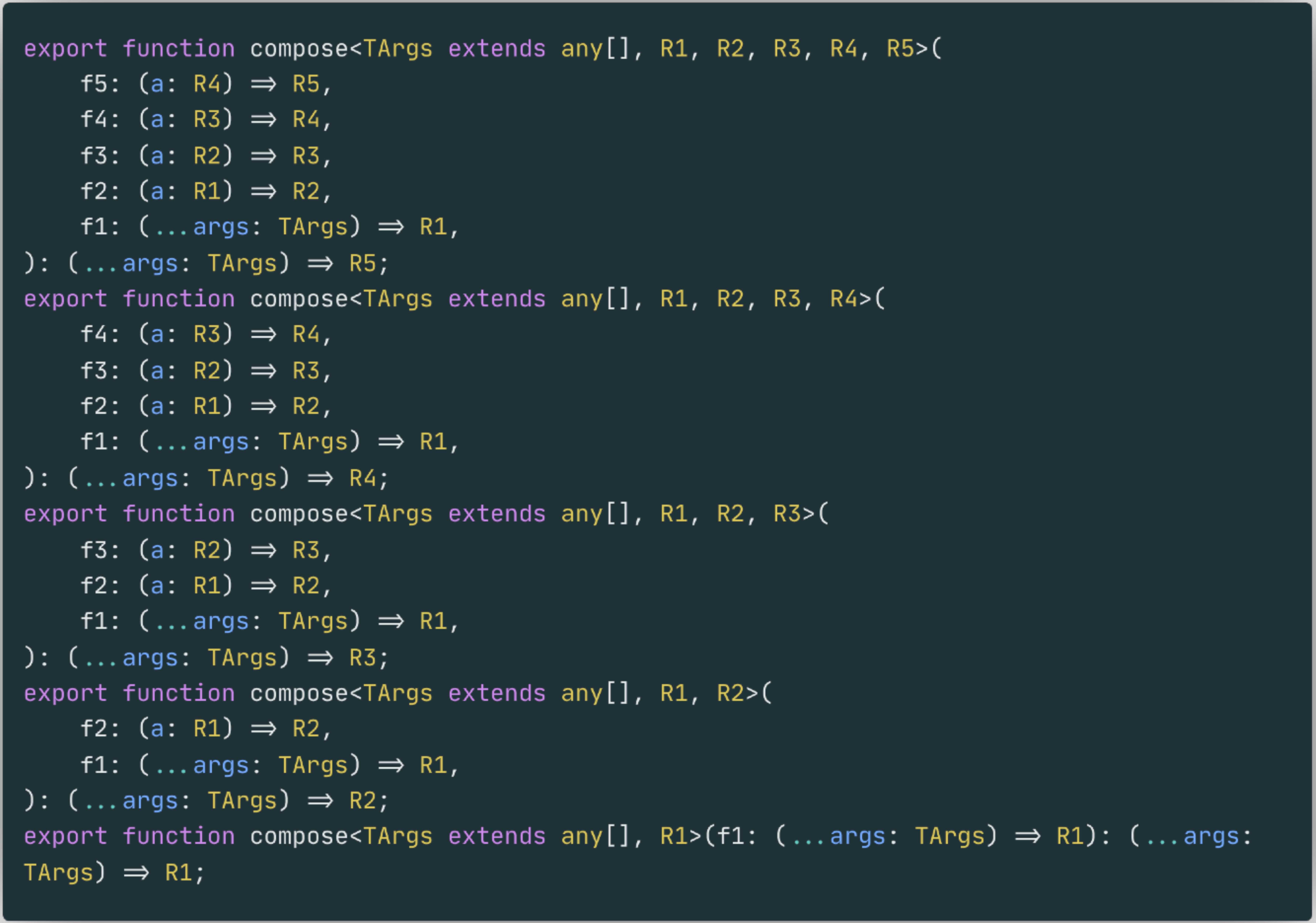


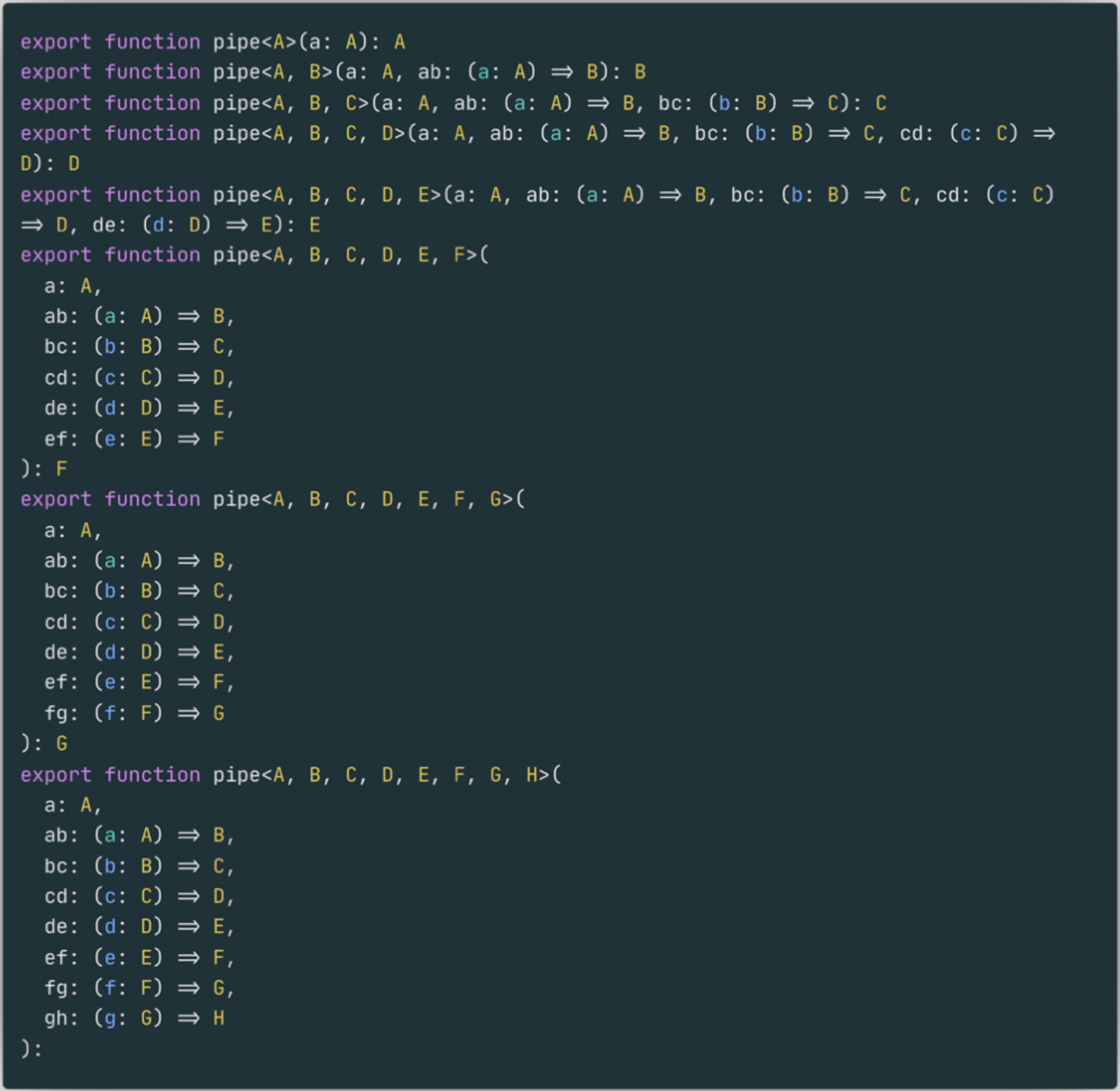
```
assign<T, U>(target: T, source: U): T & U;
assign<T, U, V>(target: T, source1: U, source2: V): T & U & V;
assign<T, U, V, W>(target: T, source1: U, source2: V, source3: W): T & U & V & W
```

Object.assign



```typescript
export function compose<TArgs extends any[], R1, R2, R3, R4, R5>(
    f5: (a: R4) ⇒ R5,
    f4: (a: R3) ⇒ R4,
    f3: (a: R2) ⇒ R3,
    f2: (a: R1) ⇒ R2,
    f1: (...args: TArgs) ⇒ R1,
): (...args: TArgs) ⇒ R5;
export function compose<TArgs extends any[], R1, R2, R3, R4>(
    f4: (a: R3) ⇒ R4,
    f3: (a: R2) ⇒ R3,
    f2: (a: R1) ⇒ R2,
    f1: (...args: TArgs) ⇒ R1,
): (...args: TArgs) ⇒ R4;
export function compose<TArgs extends any[], R1, R2, R3>(
    f3: (a: R2) ⇒ R3,
    f2: (a: R1) ⇒ R2,
    f1: (...args: TArgs) ⇒ R1,
): (...args: TArgs) ⇒ R3;
export function compose<TArgs extends any[], R1, R2>(
    f2: (a: R1) ⇒ R2,
    f1: (...args: TArgs) ⇒ R1,
): (...args: TArgs) ⇒ R2;
export function compose<TArgs extends any[], R1>(f1: (...args: TArgs) ⇒ R1): (...args:
TArgs) ⇒ R1;
```

Ramda



```
export function pipe<A>(a: A): A
export function pipe<A, B>(a: A, ab: (a: A) ⇒ B): B
export function pipe<A, B, C>(a: A, ab: (a: A) ⇒ B, bc: (b: B) ⇒ C): C
export function pipe<A, B, C, D>(a: A, ab: (a: A) ⇒ B, bc: (b: B) ⇒ C, cd: (c: C) ⇒
D): D
export function pipe<A, B, C, D, E>(a: A, ab: (a: A) ⇒ B, bc: (b: B) ⇒ C, cd: (c: C)
⇒ D, de: (d: D) ⇒ E): E
export function pipe<A, B, C, D, E, F>(
  a: A,
  ab: (a: A) ⇒ B,
  bc: (b: B) ⇒ C,
  cd: (c: C) ⇒ D,
  de: (d: D) ⇒ E,
  ef: (e: E) ⇒ F
): F
export function pipe<A, B, C, D, E, F, G>(
  a: A,
  ab: (a: A) ⇒ B,
  bc: (b: B) ⇒ C,
  cd: (c: C) ⇒ D,
  de: (d: D) ⇒ E,
  ef: (e: E) ⇒ F,
  fg: (f: F) ⇒ G
): G
export function pipe<A, B, C, D, E, F, G, H>(
  a: A,
  ab: (a: A) ⇒ B,
  bc: (b: B) ⇒ C,
  cd: (c: C) ⇒ D,
  de: (d: D) ⇒ E,
  ef: (e: E) ⇒ F,
  fg: (f: F) ⇒ G,
  gh: (g: G) ⇒ H
):
```

fp-ts

```
assign<T, U>(target: T, source: U): T & U;
assign<T, U, V>(target: T, source1: U, source2: V): T & U & V;
assign<T, U, V, W>(target: T, source1: U, source2: V, source3: W): T & U & V & W
```

```
export function compose<TArgs extends any[], R1, R2
    f5: (a: R4) ⇒ R5,
    f4: (a: R3) ⇒ R4,
    f3: (a: R2) ⇒ R3,
    f2: (a: R1) ⇒ R2,
    f1: (...args: TArgs) ⇒ R1,
): (...args: TArgs) ⇒ R5;
export function compose<TArgs extends any[], R1, R2
    f4: (a: R3) ⇒ R4,
    f3: (a: R2) ⇒ R3,
    f2: (a: R1) ⇒ R2,
    f1: (...args: TArgs) ⇒ R1,
): (...args: TArgs) ⇒ R4;
export function compose<TArgs extends any[], R1, R2
    f3: (a: R2) ⇒ R3,
    f2: (a: R1) ⇒ R2,
    f1: (...args: TArgs) ⇒ R1,
): (...args: TArgs) ⇒ R3;
export function compose<TArgs extends any[], R1, R2
    f2: (a: R1) ⇒ R2,
```

```
export function pipe<A>(a: A): A
export function pipe<A, B>(a: A, ab: (a: A) ⇒ B): B
export function pipe<A, B, C>(a: A, ab: (a: A) ⇒ B, bc: (b: B) ⇒ C): C
export function pipe<A, B, C, D>(a: A, ab: (a: A) ⇒ B, bc: (b: B) ⇒ C, cd: (c: C) ⇒
D): D
export function pipe<A, B, C, D, E>(a: A, ab: (a: A) ⇒ B, bc: (b: B) ⇒ C, cd: (c: C)
⇒ D, de: (d: D) ⇒ E): E
export function pipe<A, B, C, D, E, F>(
  a: A,
  ab: (a: A) ⇒ B,
  bc: (b: B) ⇒ C,
  cd: (c: C) ⇒ D,
  de: (d: D) ⇒ E,
  ef: (e: E) ⇒ F
): F
export function pipe<A, B, C, D, E, F, G>(
  a: A,
  ab: (a: A) ⇒ B,
  bc: (b: B) ⇒ C,
  cd: (c: C) ⇒ D,
  de: (d: D) ⇒ E,
  ef: (e: E) ⇒ F,
  fg: (f: F) ⇒ G
): G
export function pipe<A, B, C, D, E, F, G, H>(
  a: A,
  ab: (a: A) ⇒ B,
  bc: (b: B) ⇒ C,
  cd: (c: C) ⇒ D,
  de: (d: D) ⇒ E
```

fp-ts

```
const schema = f.shape({
  name: f(string(), required()),
  version: f(pattern(/\d+\.\d+.\d+/)),
  workspaces: f.array(f(string())),
  repository: f.shape({
    type: f(string(), required(), oneOf(['git', 'vcs']), nullable()),
    url: f(string(), pattern(urlRegex))
  })
})

/*
const schema: Fuji<"shape", {
    name: Fuji<"string" | "required", string>;
    version: Fuji<"pattern", string>;
    workspaces: Fuji<"array-of", string[]>;
    repository: Fuji<"shape", {
        type: Fuji<...>;
        url: Fuji<...>;
    }>;
}>
*/
```