ValueScript

A dialect of TypeScript with Value Semantics

Andrew Morris

https://ValueScript.org

voltrevo/ValueScript

TypeScript

```
export default function main() {
  let pirateEnabled = false;

  function greet() {
    if (!pirateEnabled) {
      return "Hi";
    }

    return "Ahoy";
  }

  function enablePirate() {
    pirateEnabled = true;
    return "Done";
  }

  return [
    greet(),        // Hi
    enablePirate(), // Done
    greet(),        // Ahoy
  ];
}
```

TypeScript

```typescript
export default function main() {
  let pirateEnabled = false;

  function greet() {
    if (!pirateEnabled) {
      return "Hi";
    }

    return "Ahoy";
  }

  function enablePirate() {
    pirateEnabled = true;
    return "Done";
  }

  return [
    greet(),        // Hi
    enablePirate(), // Done
    greet(),        // Ahoy
  ];
}
```

Rust

```rust
fn main() {
    let mut pirate_enabled = false;

    let greet = || {
        if !pirate_enabled {
            "Hi"
        } else {
            "Ahoy"
        }
    };

    let mut enable_pirate = || {
        pirate_enabled = true;
        "Done"
    };

    dbg!(vec![
        greet(),
        enable_pirate(),
        greet(),
    ]);
}
```

cannot borrow `pirate_enabled`
as mutable because it is also
borrowed as immutable

ValueScript

```
export default function main() {
  let pirateEnabled = false;

  function greet() {
    if (!pirateEnabled) {
      return "Hi";
    }

    return "Ahoy";
  }

  function enablePirate() {
    pirateEnabled = true;
    return "Done";
  }

  return [
    greet(),
    enablePirate(),
    greet(),
  ];
}
```

Cannot mutate captured variable `pirateEnabled`

Rust

```
fn main() {
    let mut pirate_enabled = false;

    let greet = || {
        if !pirate_enabled {
            "Hi"
        } else {
            "Ahoy"
        }
    };

    let mut enable_pirate = || {
        pirate_enabled = true;
        "Done"
    };

    dbg!(vec![
        greet(),
        enable_pirate(),
        greet(),
    ]);
}
```

cannot borrow `pirate_enabled` as mutable because it is also borrowed as immutable

ValueScript

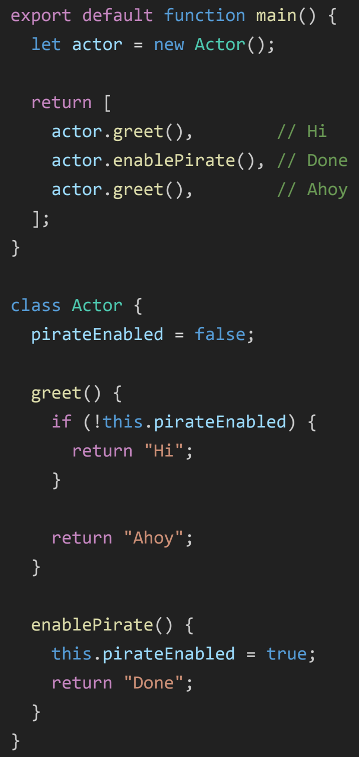
```
export default function main() {
  let actor = new Actor();

  return [
    actor.greet(),        // Hi
    actor.enablePirate(), // Done
    actor.greet(),        // Ahoy
  ];
}

class Actor {
  pirateEnabled = false;

  greet() {
    if (!this.pirateEnabled) {
      return "Hi";
    }

    return "Ahoy";
  }

  enablePirate() {
    this.pirateEnabled = true;
    return "Done";
  }
}
```

Rust

```
fn main() {
    let mut pirate_enabled = false;

    let greet = || {
        if !pirate_enabled {
            "Hi"
        } else {
            "Ahoy"
        }
    };

    let mut enable_pirate = || {
        pirate_enabled = true;
        "Done"
    };

    dbg!(vec![
        greet(),
        enable_pirate(),
        greet(),
    ]);
}
```

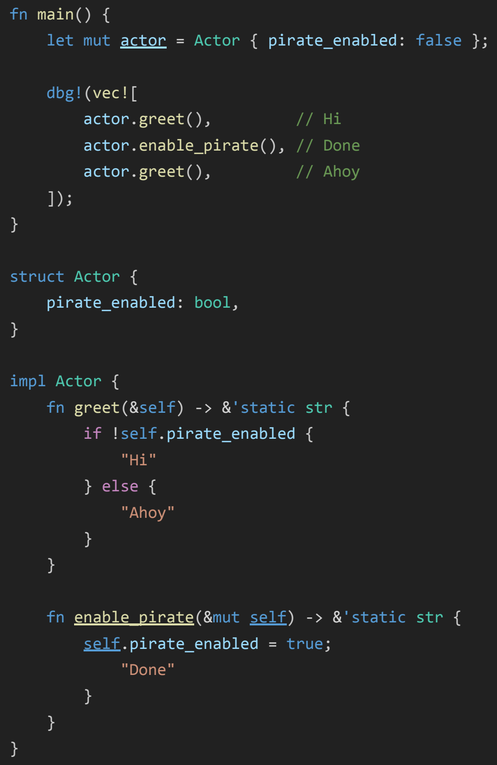
cannot borrow `pirate_enabled`
as mutable because it is also
borrowed as immutable

ValueScript



```
export default function main() {
  let actor = new Actor();

  return [
    actor.greet(),        // Hi
    actor.enablePirate(), // Done
    actor.greet(),        // Ahoy
  ];
}

class Actor {
  pirateEnabled = false;

  greet() {
    if (!this.pirateEnabled) {
      return "Hi";
    }

    return "Ahoy";
  }

  enablePirate() {
    this.pirateEnabled = true;
    return "Done";
  }
}
```

Rust



```
fn main() {
    let mut actor = Actor { pirate_enabled: false };

    dbg!(vec![
        actor.greet(),         // Hi
        actor.enable_pirate(), // Done
        actor.greet(),         // Ahoy
    ]);
}

struct Actor {
    pirate_enabled: bool,
}

impl Actor {
    fn greet(&self) -> &'static str {
        if !self.pirate_enabled {
            "Hi"
        } else {
            "Ahoy"
        }
    }

    fn enable_pirate(&mut self) -> &'static str {
        self.pirate_enabled = true;
            "Done"
        }
    }
}
```

Value Semantics

```
export default function main() {
  const leftBowl = ["apple", "mango"];

  let rightBowl = leftBowl;
  rightBowl.push("peach");

  return leftBowl.includes("peach");
  // JavaScript: true
  // ValueScript: false
}
```

Value Semantics

```
export default function main() {
  const leftBowl = ["apple", "mango"];

  let rightBowl = leftBowl;
  rightBowl.push("peach");

  return leftBowl.includes("peach");
  // JavaScript: true
  // ValueScript: false
}
```

JavaScript

```
leftBowl  →  ["apple", "mango"]
```

ValueScript

```
leftBowl  →  ["apple", "mango"]
```

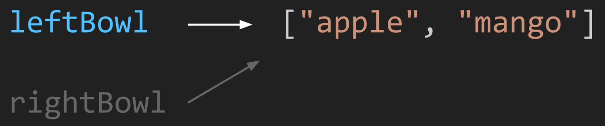
Rust

```
leftBowl  →  ["apple", "mango"]
```

Value Semantics

```
export default function main() {
  const leftBowl = ["apple", "mango"];

  let rightBowl = leftBowl;
  rightBowl.push("peach");

  return leftBowl.includes("peach");
  // JavaScript: true
  // ValueScript: false
}
```

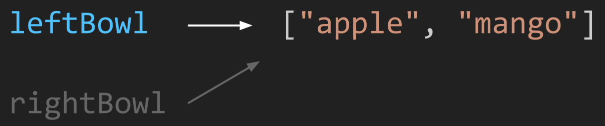
JavaScript



leftBowl → ["apple", "mango"]

rightBowl

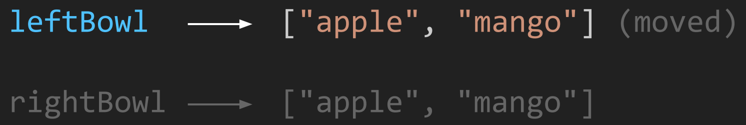
ValueScript



leftBowl → ["apple", "mango"]

rightBowl

Rust

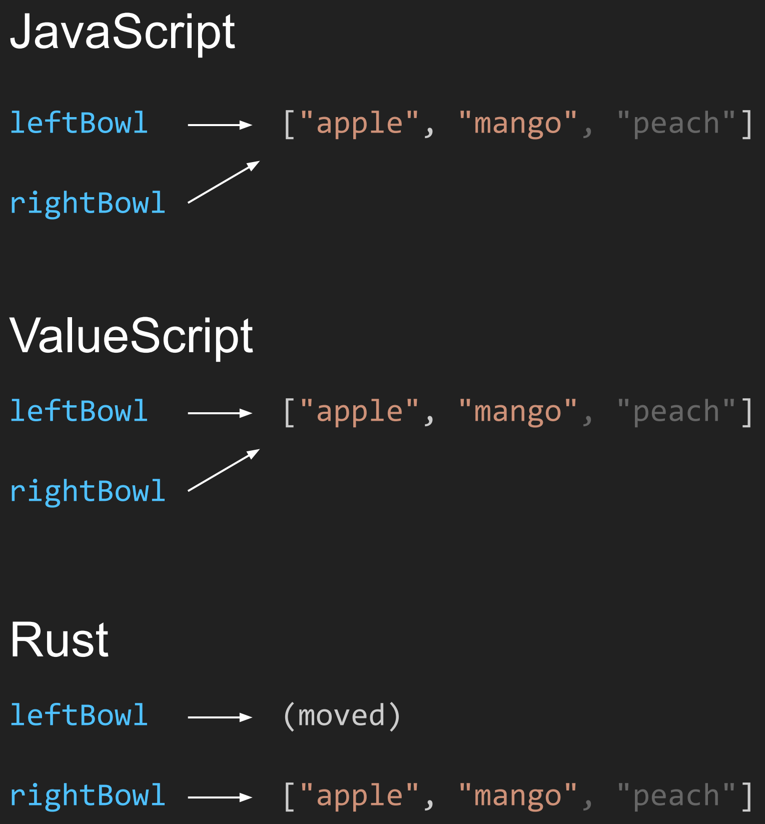


leftBowl → ["apple", "mango"] (moved)

rightBowl → ["apple", "mango"]

Value Semantics

```
export default function main() {
  const leftBowl = ["apple", "mango"];

  let rightBowl = leftBowl;
  rightBowl.push("peach");

  return leftBowl.includes("peach");
  // JavaScript: true
  // ValueScript: false
}
```



JavaScript

leftBowl → ["apple", "mango", "peach"]

rightBowl

ValueScript

leftBowl → ["apple", "mango", "peach"]

rightBowl

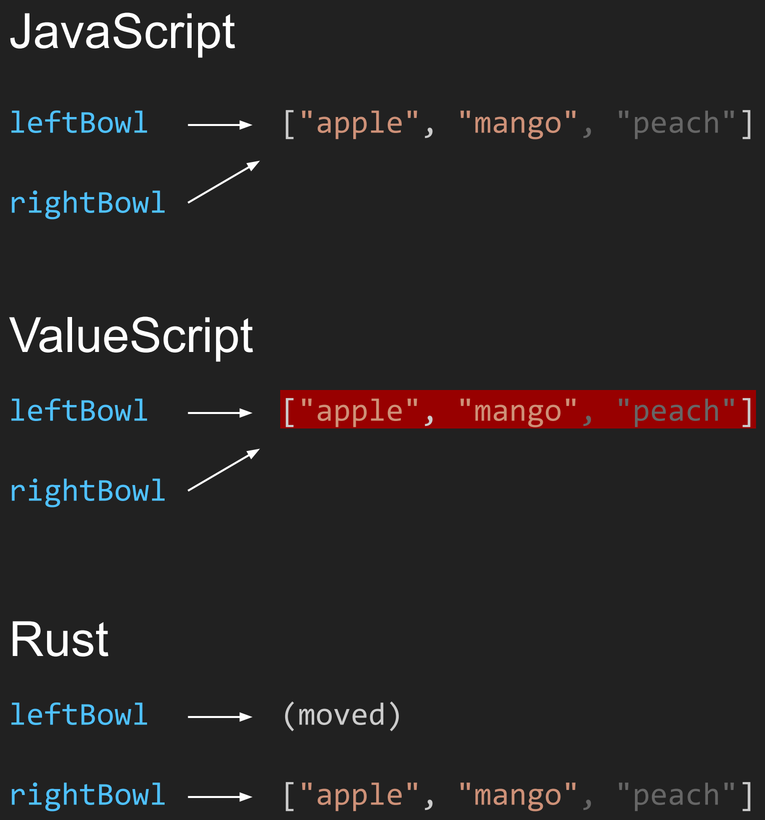
Rust

leftBowl → (moved)

rightBowl → ["apple", "mango", "peach"]

Value Semantics

```
export default function main() {
  const leftBowl = ["apple", "mango"];

  let rightBowl = leftBowl;
  rightBowl.push("peach");

  return leftBowl.includes("peach");
  // JavaScript: true
  // ValueScript: false
}
```



JavaScript

leftBowl → ["apple", "mango", "peach"]

rightBowl

ValueScript

leftBowl → ["apple", "mango", "peach"]

rightBowl

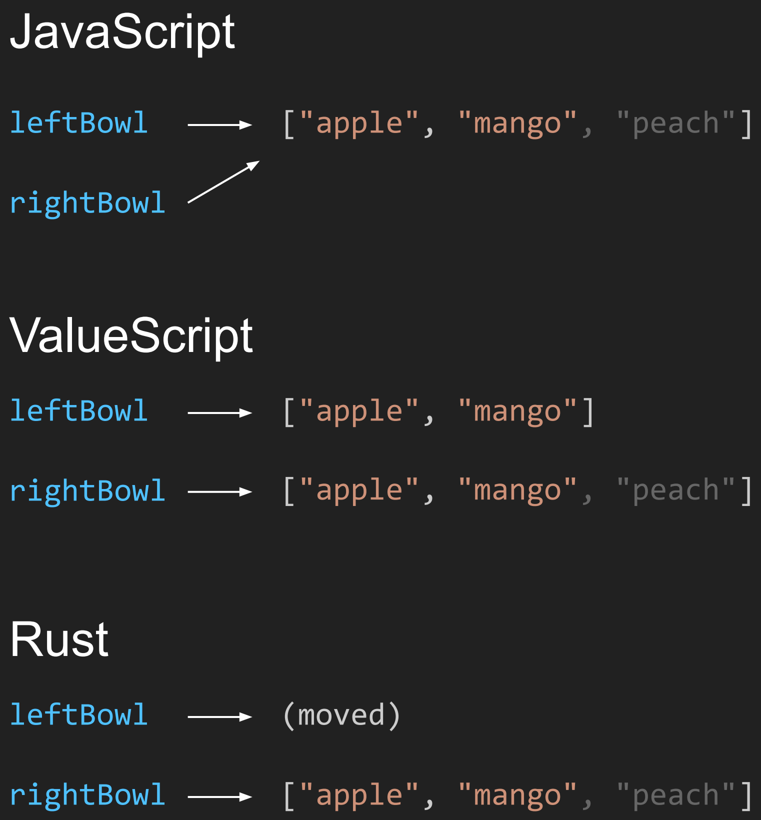
Rust

leftBowl → (moved)

rightBowl → ["apple", "mango", "peach"]

Value Semantics

```
export default function main() {
  const leftBowl = ["apple", "mango"];

  let rightBowl = leftBowl;
  rightBowl.push("peach");

  return leftBowl.includes("peach");
  // JavaScript: true
  // ValueScript: false
}
```



JavaScript

leftBowl → ["apple", "mango", "peach"]

rightBowl ↗

ValueScript

leftBowl → ["apple", "mango"]

rightBowl → ["apple", "mango", "peach"]

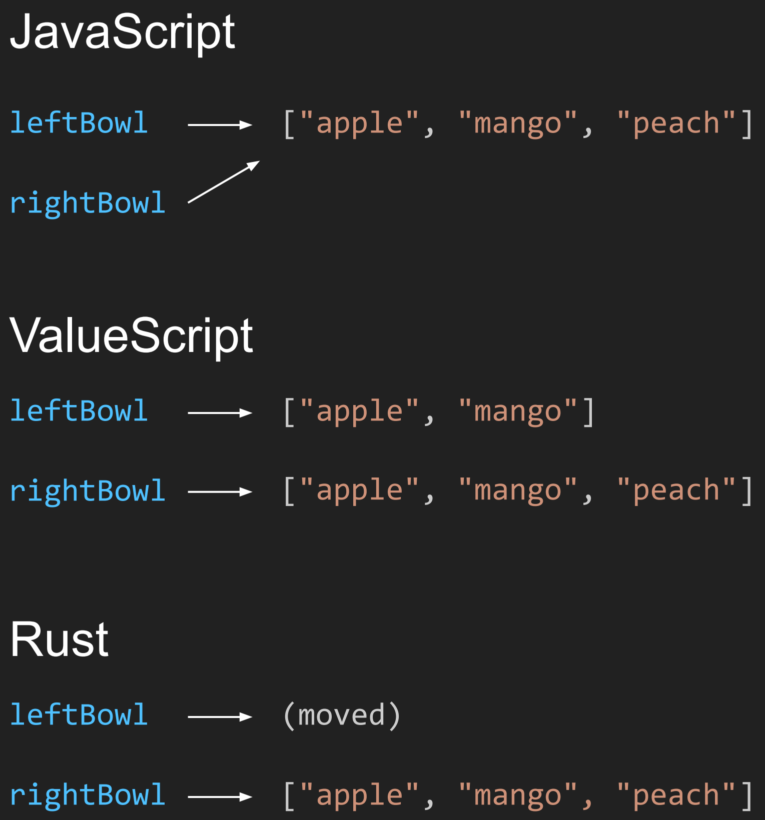
Rust

leftBowl → (moved)

rightBowl → ["apple", "mango", "peach"]

Value Semantics

```
export default function main() {
  const leftBowl = ["apple", "mango"];

  let rightBowl = leftBowl;
  rightBowl.push("peach");

  return leftBowl.includes("peach");
  // JavaScript: true
  // ValueScript: false
}
```



JavaScript

leftBowl → ["apple", "mango", "peach"]

rightBowl ↗

ValueScript

leftBowl → ["apple", "mango"]

rightBowl → ["apple", "mango", "peach"]

Rust

leftBowl → (moved)

rightBowl → ["apple", "mango", "peach"]

Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

Value Semantics

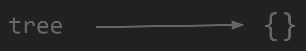
```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

```
class BinaryTree<T extends NotNullish> {
  left?: BinaryTree<T>;
  value?: T;
  right?: BinaryTree<T>;

  insert(newValue: T) {
    if (this.value === undefined) {
      this.value = newValue;
      return;
    }

    if (newValue < this.value) {
      this.left ??= new BinaryTree();
      this.left.insert(newValue);
    } else {
      this.right ??= new BinaryTree();
      this.right.insert(newValue);
    }
  }
}
```

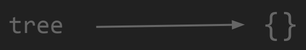
Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript



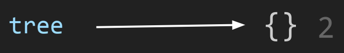
ValueScript



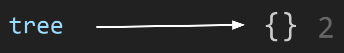
Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript



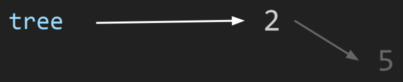
ValueScript



Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript

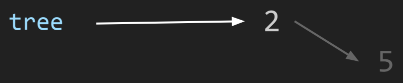


tree

2

5

ValueScript



tree

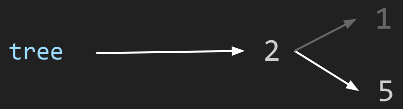
2

5

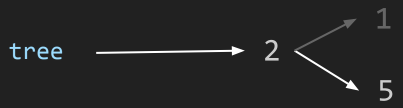
Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript



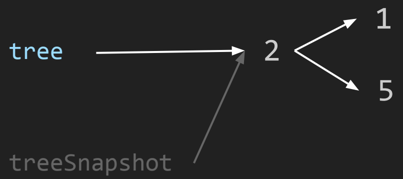
ValueScript



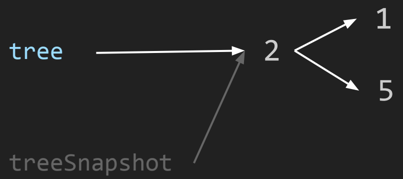
Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript



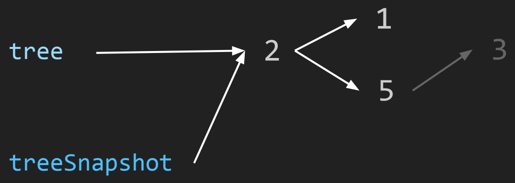
ValueScript



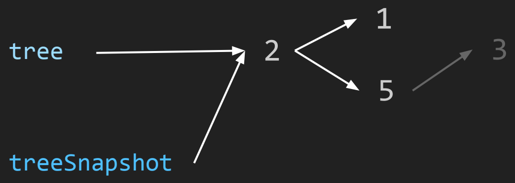
Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript



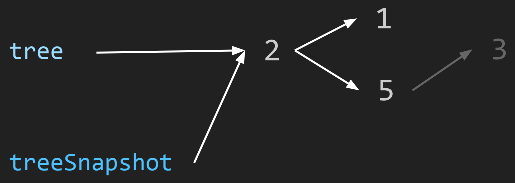
ValueScript



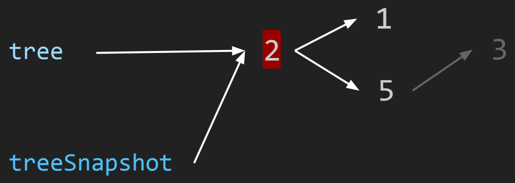
Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript



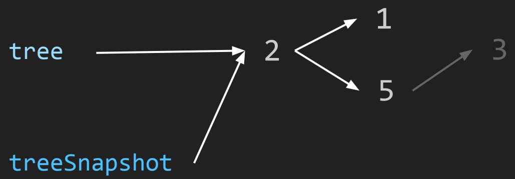
ValueScript



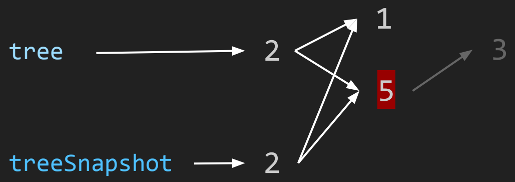
Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript



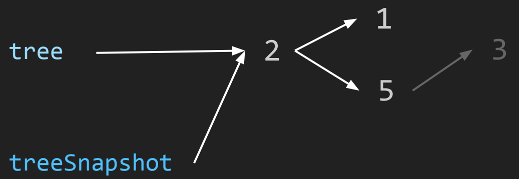
ValueScript



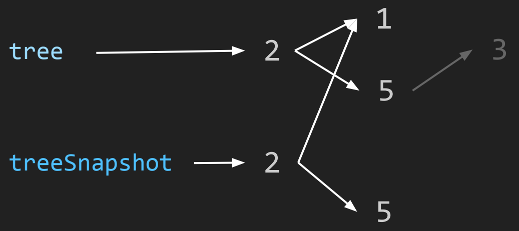
Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript



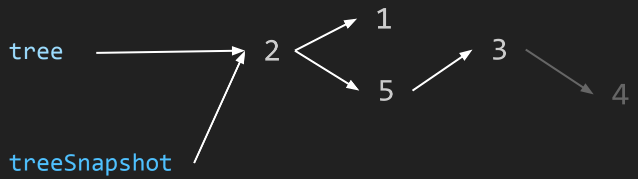
ValueScript



Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript



tree

treeSnapshot

2

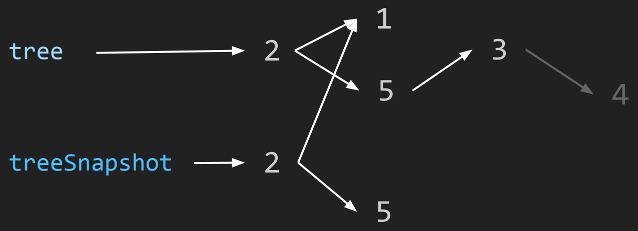
1

5

3

4

ValueScript



tree

treeSnapshot

2

1

5

3

4

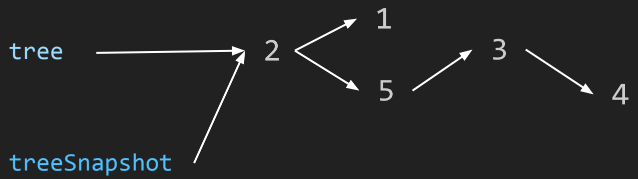
2

5

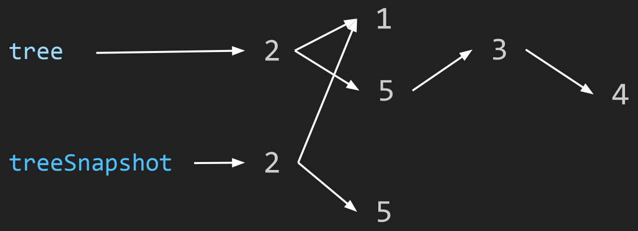
Value Semantics

```
import { BinaryTree } from "../lib/mod.ts";

export default function main() {
  let tree = new BinaryTree<number>();

  tree.insert(2);
  tree.insert(5);
  tree.insert(1);

  const treeSnapshot = tree;

  tree.insert(3);
  tree.insert(4);

  return [[...treeSnapshot], [...tree]];
  // JavaScript: [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  // ValueScript: [[1, 2, 5], [1, 2, 3, 4, 5]]
}
```

JavaScript



ValueScript



Transactional Try Blocks

```javascript
export default function () {
  let x = 0;

  try {
    x++;
    throw new Error("boom");
  } catch {}

  return x;
  // JavaScript: 1
  // ValueScript: 0
}
```

Playground

Questions?

https://ValueScript.org

voltrevo/ValueScript