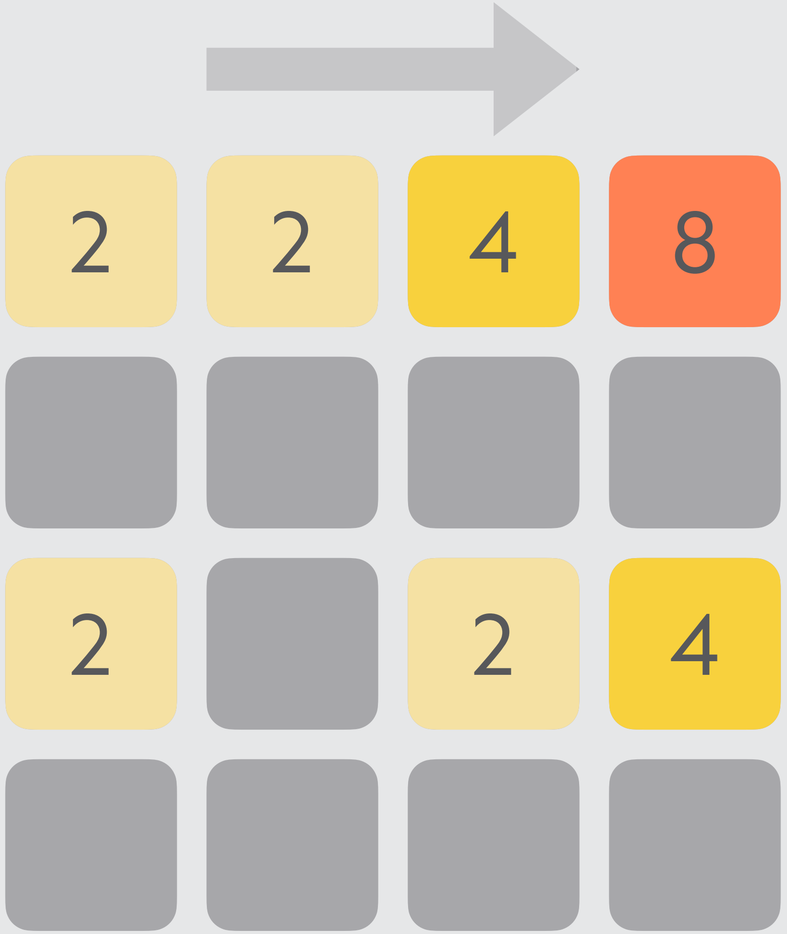
SYMMETRIC PROGRAMMING

Matija Pretnar
University of Ljubljana

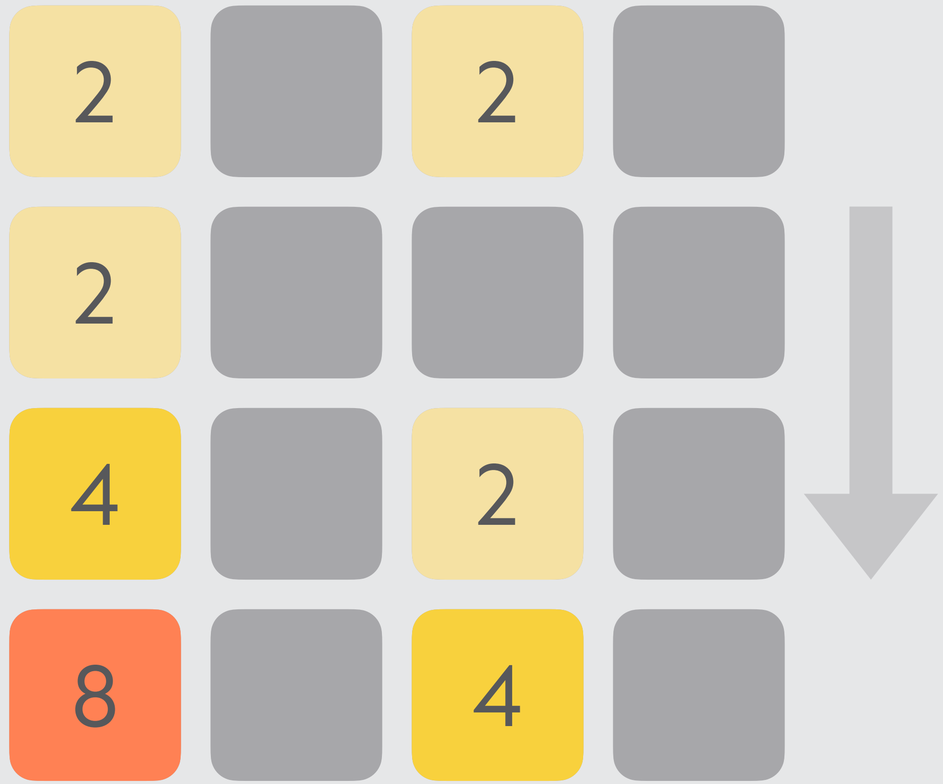
Ohad Kammar
University of Edinburgh

In 2048, player **merges tiles** of by sliding them **together**

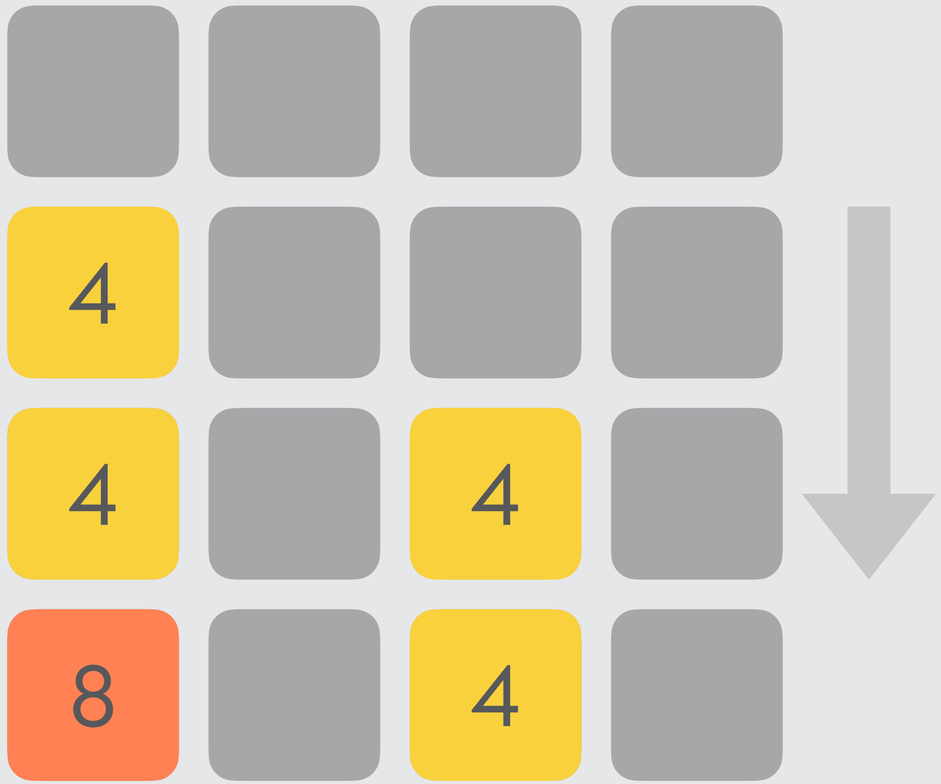


In 2048, player **merges tiles** of by sliding them **together**

Merging in other directions is **symmetric**



Merging in other directions is **symmetric**



We can employ a **clever trick**

We can employ a **clever trick**

```
// Get the vector representing the chosen direction
GameManager.prototype.getVector = function (direction) {
  // Vectors representing tile movement
  var map = {
    0: { x: 0,  y: -1 }, // Up
    1: { x: 1,  y: 0 },  // Right
    2: { x: 0,  y: 1 },  // Down
    3: { x: -1, y: 0 }   // Left
  };

  return map[direction];
};
```

We can employ a **clever trick**

```
// Get the vector representing the chosen direction
GameManager.prototype.getVector = function (direction) {
  // Vectors representing tile movement
  var map = {
    0: { x: 0,  y: -1 }, // up
    1: { x: 1,  y: 0 },  // right
    2: { x: 0,  y: 1 },  // down
    3: { x: -1, y: 0 }   // left
  };

  return map[direction];
};
```

```
// 0: up, 1: right, 2: down, 3: left
var vector     = this.getVector(direction);
var traversals = this.buildTraversals(vector);
// Traverse the grid in the right direction and move tiles
traversals.x.forEach(function (x) {
  traversals.y.forEach(function (y) {
    cell = { x: x, y: y };
    tile = self.grid.cellContent(cell);

    if (tile) {
      var positions = self.findFarthestPosition(cell, vector);
      var next      = self.grid.cellContent(positions.next);

      // Only one merger per row traversal?
      if (next && next.value === tile.value && !next.mergedFrom) {
        var merged = new Tile(positions.next, tile.value * 2);
        merged.mergedFrom = [tile, next];
...
```

We can employ a **clever trick**

```
// Get the vector representing the chosen direction
GameManager.prototype.getVector = function (direction) {
  // Vectors representing tile movement
  var map
    0: {
    1: {
    2: {
    3: {
  };

  return
};
```

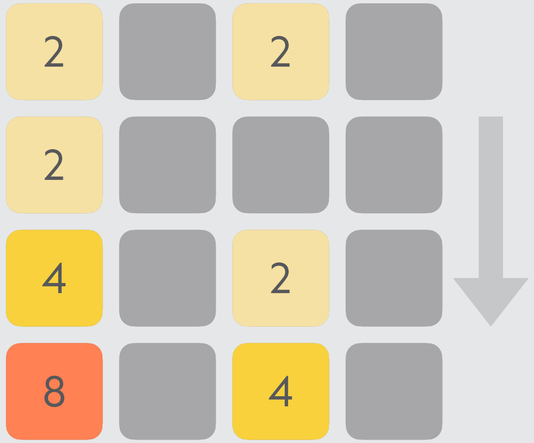
```
// 0: up, 1: right, 2: down, 3: left
var vector     = this.getVector(direction);
var traversals = this.buildTraversals(vector);
// Traverse the grid in the right direction and move tiles
traversals.x.forEach(function (x) {
  traversals.y.forEach(function (y) {
    cell = { x: x, y: y };
    tile = self.grid.cellContent(cell);

    if (tile) {
      var positions = self.findFarthestPosition(cell, vector);
      var next      = self.grid.cellContent(positions.next);

      // Only one merger per row traversal?
      if (next && next.value === tile.value && !next.mergedFrom) {
        var merged = new Tile(positions.next, tile.value * 2);
        merged.mergedFrom = [tile, next];
...
```

We can employ a **clever trick**

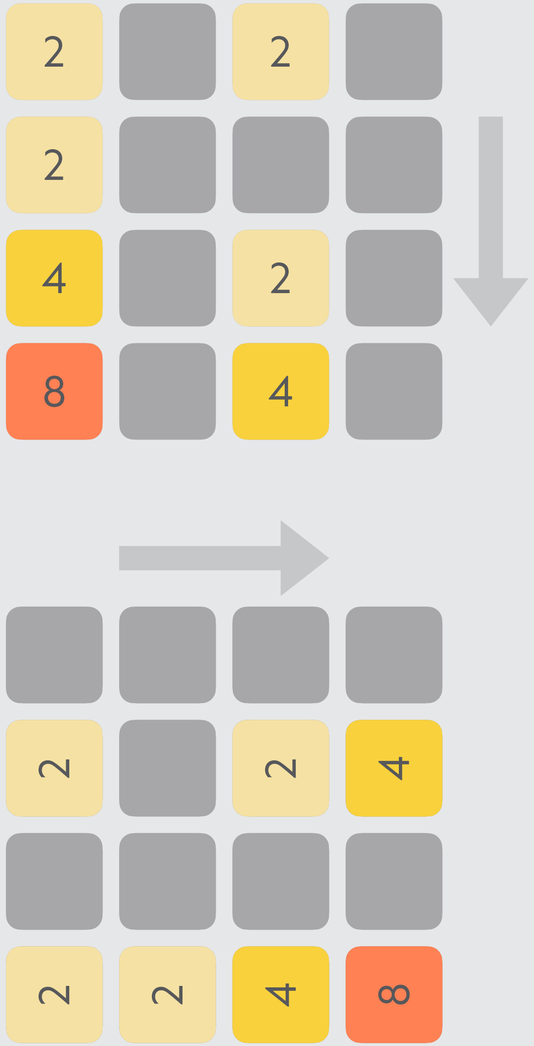
```
// Get the vector representing the chosen direction
GameManager.prototype.getVector = function (direction) {
  // Vectors representing tile movement
  var map
    0: {
    1: {
    2: {
    3: {
  };

  return
};
```

```
// 0: up, 1: right, 2: down, 3: left
var vector     = this.getVector(direction);
var traversals = this.buildTraversals(vector);
// Traverse the grid in the right direction and move tiles
traversals.x.forEach(function (x) {
  traversals.y.forEach(function (y) {
    cell = { x: x, y: y };
    tile = self.grid.cellContent(cell);

    if (tile) {
      var positions = self.findFarthestPosition(cell, vector);
      var next      = self.grid.cellContent(positions.next);

      // Only one merger per row traversal?
      if (next && next.value === tile.value && !next.mergedFrom) {
        var merged = new Tile(positions.next, tile.value * 2);
        merged.mergedFrom = [tile, next];
...
```

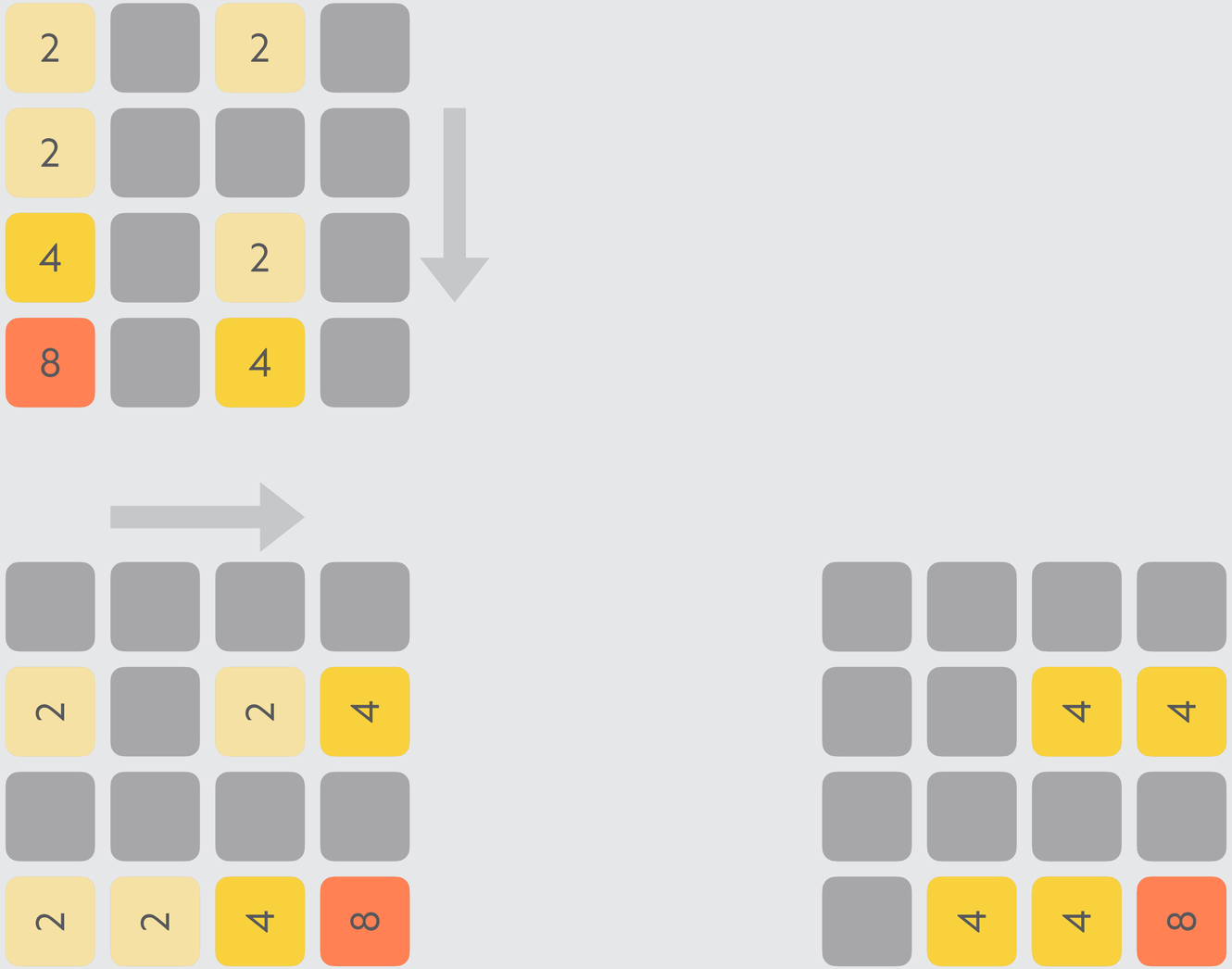
Or we can use a **simple** and **inefficient** method



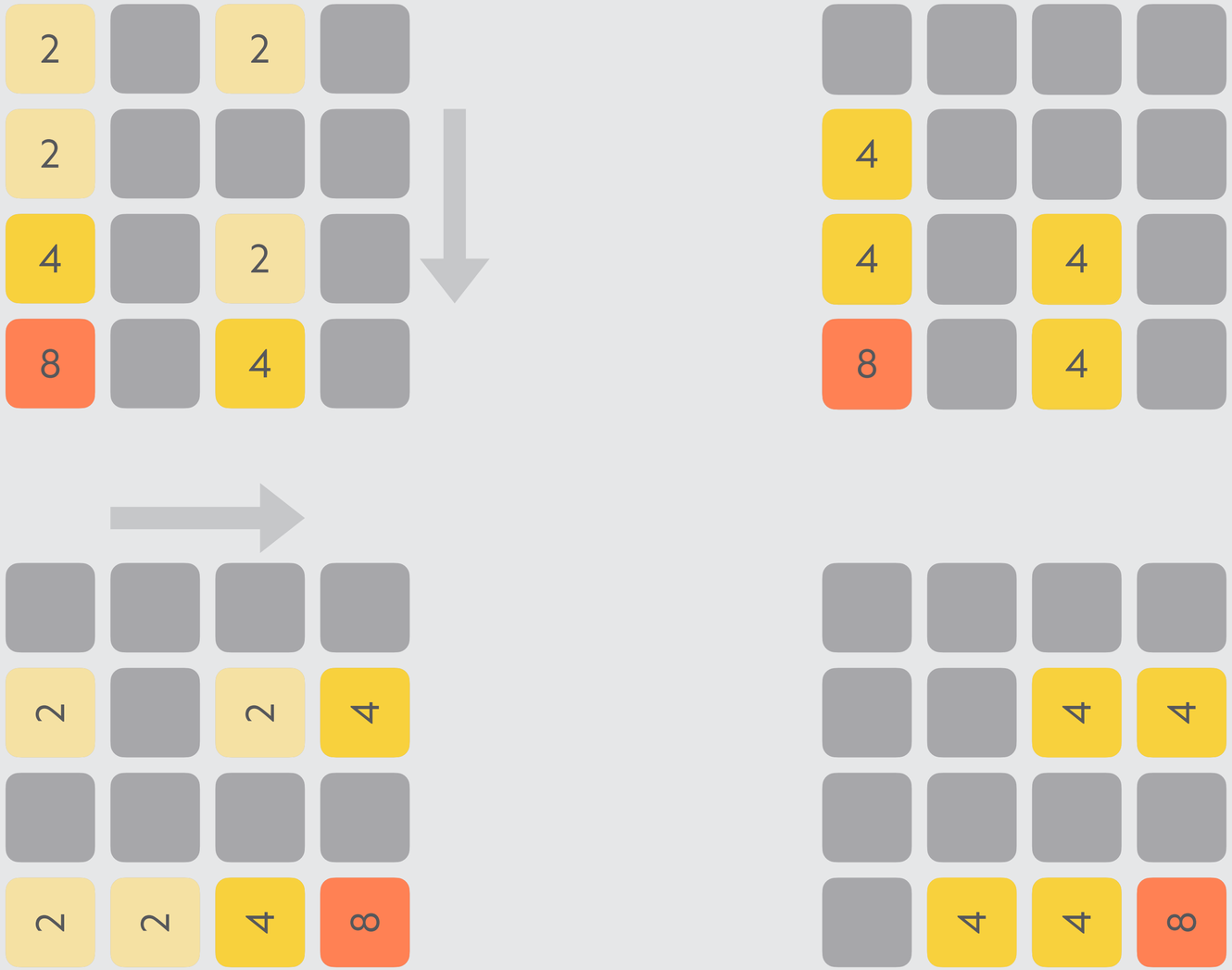
Or we can use a **simple** and **inefficient** method



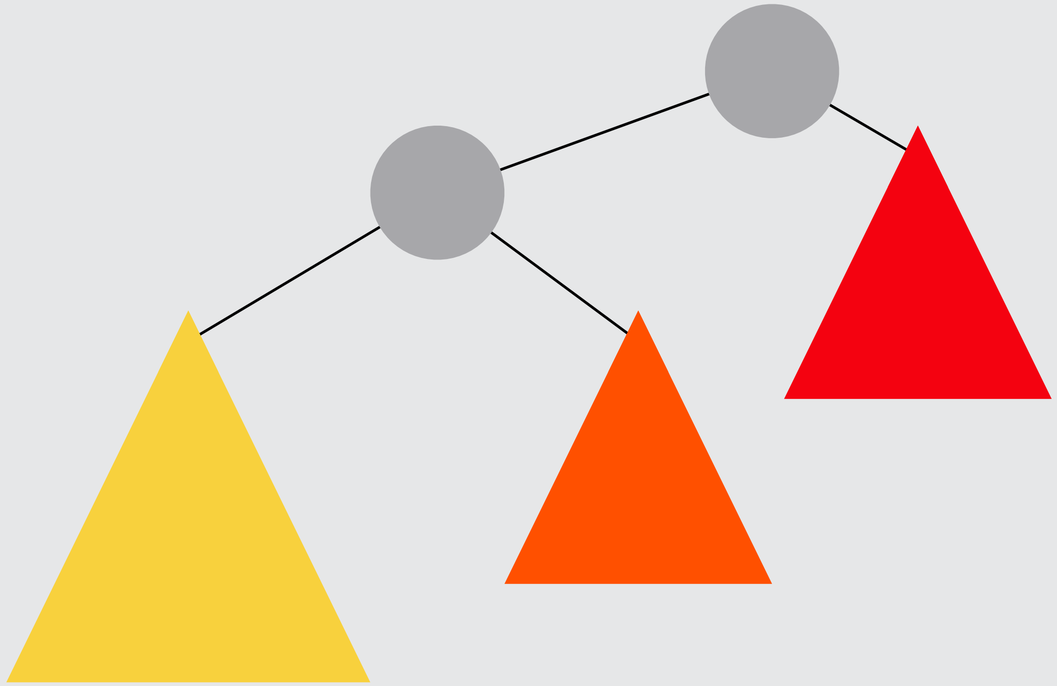
Or we can use a **simple** and **inefficient** method



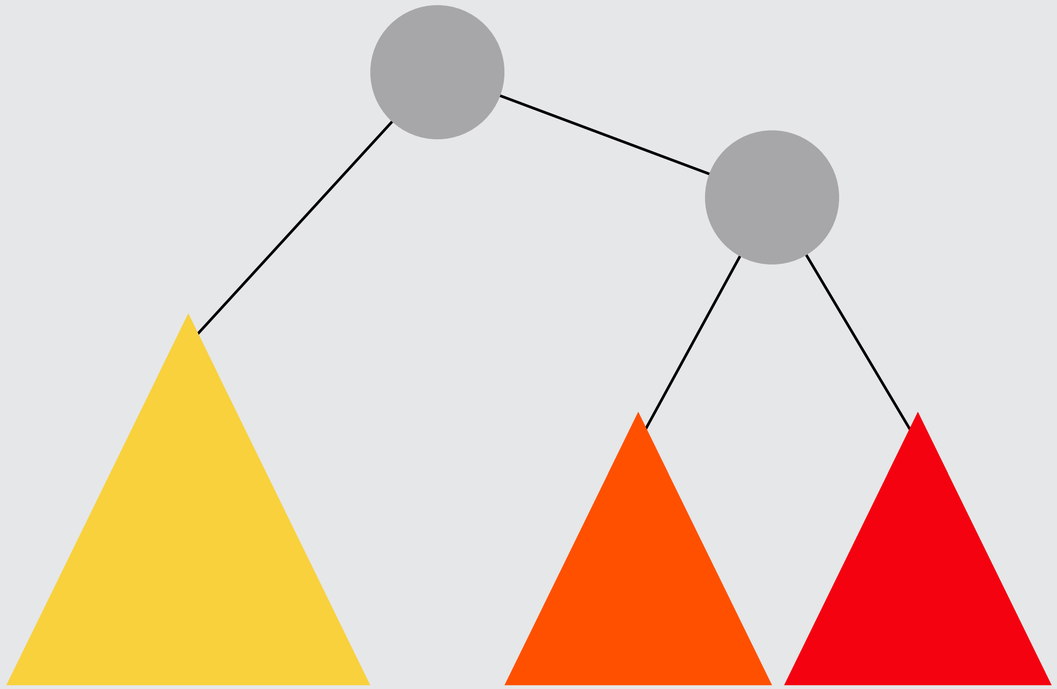
Or we can use a **simple** and **inefficient** method



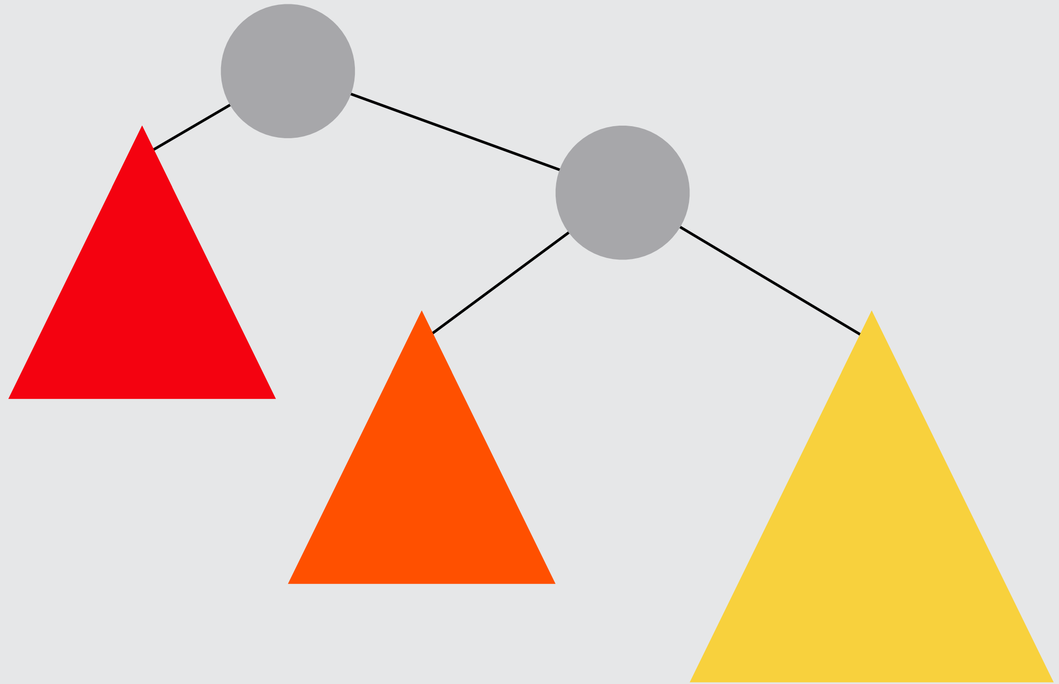
In **self-balancing** search trees, we employ left rotations



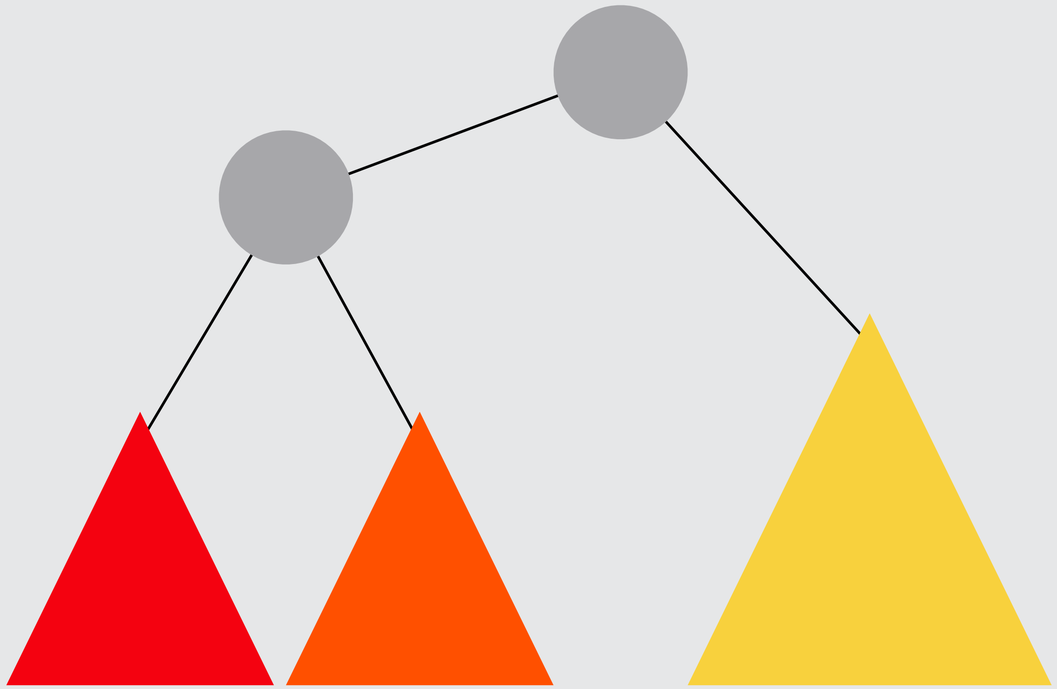
In **self-balancing** search trees, we employ left rotations



Rotating in the other direction is **symmetric**



Rotating in the other direction is **symmetric**



We can employ a **clever trick**

We can employ a **clever trick**

```
Node* rotate_subtree(Tree* tree, Node* sub, Direction dir) {
    Node* sub_parent = sub->parent;
    // 1 - dir is the opposite direction
    Node* new_root = sub->child[1 - dir];
    Node* new_child = new_root->child[dir];

    sub->child[1 - dir] = new_child;

    if (new_child) {
        new_child->parent = sub;
    }

    new_root->child[dir] = sub;

    new_root->parent = sub_parent;
    sub->parent = new_root;
    if (sub_parent) {
        sub_parent->child[sub == sub_parent->right] = new_root;
    } else {
        tree->root = new_root;
    }

    return new_root;
}
```

We can employ a **clever trick**

```
Node* rotate_subtree(Tree* tree, Node* sub, Direction dir) {
    Node* sub_parent = sub->parent;
    // 1 - dir is the opposite direction
    Node* new_root = sub->child[1 - dir];
    Node* new_child = new_root->child[dir];

    sub->child[1 - dir] = new_child;

    if (new_child) {
        new_child->parent = sub;
    }

    new_root->child[dir] = sub;

    new_root->parent = sub_parent;
    sub->parent = new_root;
    if (sub_parent) {
        sub_parent->child[sub == sub_parent->right] = new_root;
    } else {
        tree->root = new_root;
    }

    return new_root;
}
```

We can employ a **clever trick**

```
Node* rotate_subtree(Tree* tree, Node* sub, Direction dir) {
    Node* sub_parent = sub->parent;
    // 1 - dir is the opposite direction
    Node* new_root = sub->child[1 - dir];
    Node* new_child = new_root->child[dir];
```

```
// red-black tree node
typedef struct Node {
    struct Node* parent; // null for the root node
    union {
        // Union so we can use ->left/->right or ->child[0]/->child[1]
        struct {
            struct Node* left;
            struct Node* right;
        };
        struct Node* child[2];
    };
    Color color;
    int key;
} Node;
```

```
    return new_root;
}
```

We can employ a **clever trick**

```
Node* rotate_subtree(Tree* tree, Node* sub, Direction dir) {
    Node* sub_parent = sub->parent;
    // 1 - dir is the opposite direction
    Node* new_root = sub->child[1 - dir];
    Node* new_child = new_root->child[dir];
```

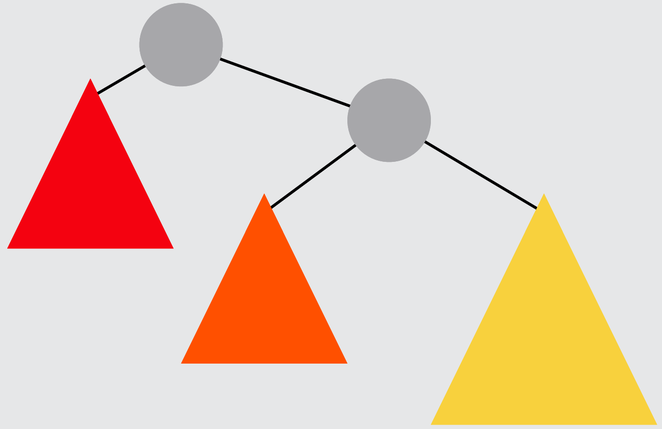
```
// red-black tree node
typedef struct Node {
    struct Node* parent; // null for the root node
    union {
        // Union so we can use ->left/->right or ->child[0]/->child[1]
        struct {
            struct Node* left;
            struct Node* right;
        };
        struct Node* child[2];
    };
    Color color;
    int key;
} Node;
```

```
    return new_root;
}
```

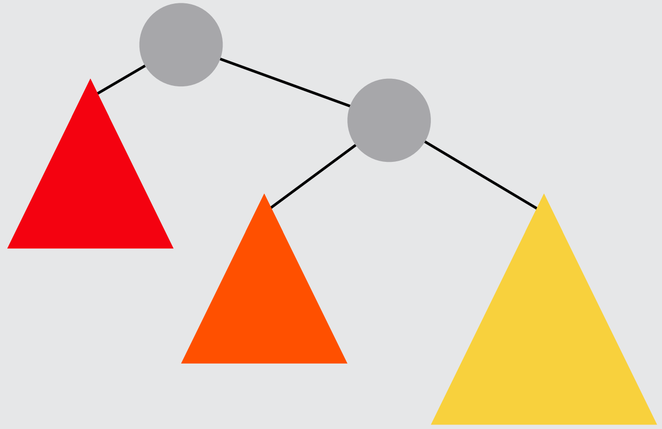
Or we can **duplicate** the code

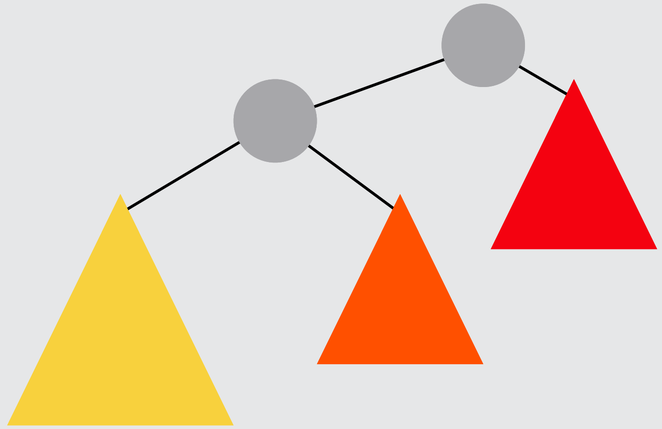
```
let rotate_left = function
  | Node (l, x, Node (rl, y, rr)) ->
      Node (Node (l, x, rl), y, rr)
  | _ -> invalid_arg "rotate_left"

let rotate_right = function
  | Node (Node (ll, y, lr), x, r) ->
      Node (ll, y, Node (lr, x, r))
  | _ -> invalid_arg "rotate_right"
```

Or we can use a **simple** and **inefficient** method

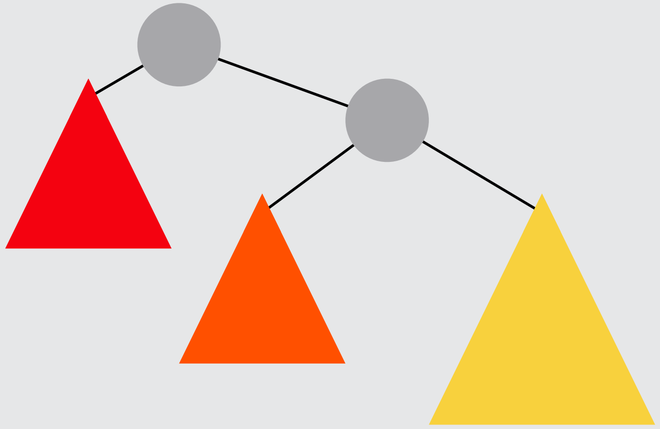


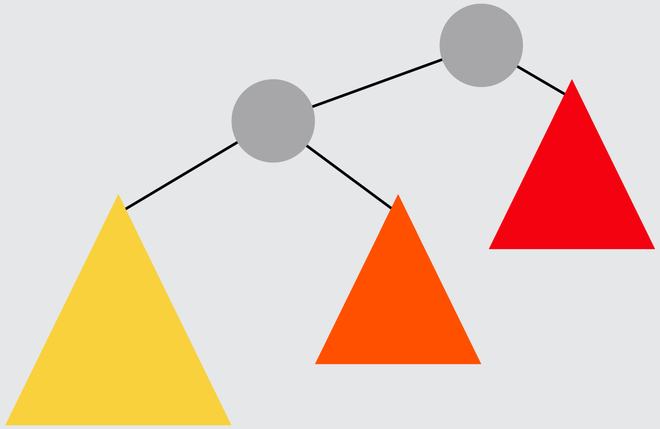
Or we can use a **simple** and **inefficient** method

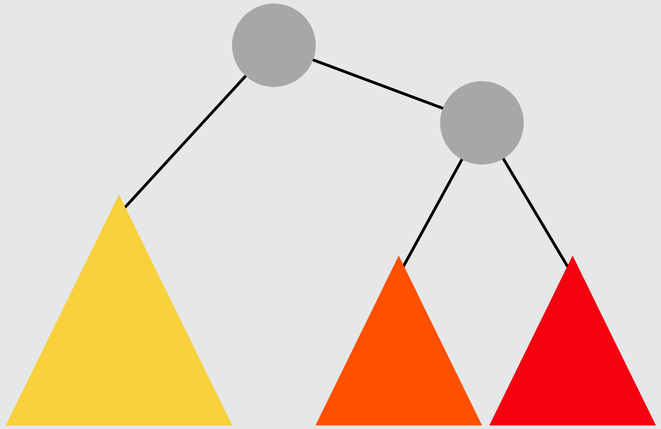




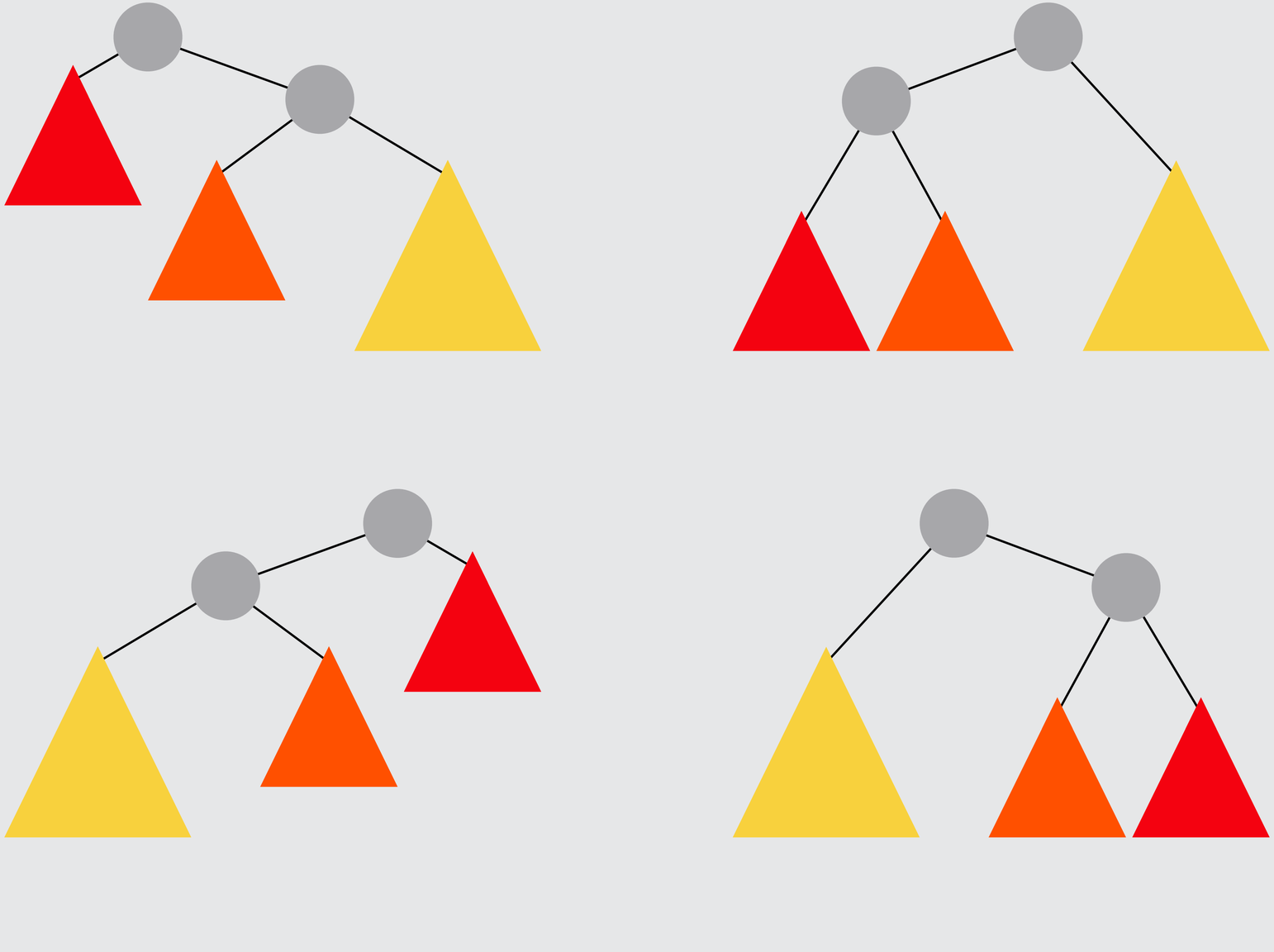
Or we can use a **simple** and **inefficient** method







Or we can use a **simple** and **inefficient** method



Mathematical proofs **often** feature **wlog assumptions**

Theorem (Schur's inequality).

For all $x, y, z \geq 0$ *and* $n \in \mathbb{N}$*, we have*

$$x^n(x-y)(x-z) + y^n(y-x)(y-z) + z^n(z-x)(z-y) \geq 0$$

Mathematical proofs **often** feature **wlog assumptions**

Theorem (Schur's inequality).

For all $x, y, z \geq 0$ *and* $n \in \mathbb{N}$*, we have*

$$x^n(x-y)(x-z) + y^n(y-x)(y-z) + z^n(z-x)(z-y) \geq 0$$

Proof.

Mathematical proofs **often** feature **wlog assumptions**

Theorem (Schur's inequality).

For all $x, y, z \geq 0$ *and* $n \in \mathbb{N}$*, we have*

$$x^n(x-y)(x-z) + y^n(y-x)(y-z) + z^n(z-x)(z-y) \geq 0$$

Proof.

Without loss of generality, assume $x \geq y \geq z$ and rearrange:

Mathematical proofs **often** feature **wlog assumptions**

Theorem (Schur's inequality).

For all $x, y, z \geq 0$ and $n \in \mathbb{N}$, we have

$$x^n(x-y)(x-z) + y^n(y-x)(y-z) + z^n(z-x)(z-y) \geq 0$$

Proof.

Without loss of generality, assume $x \geq y \geq z$ and rearrange:

$$(x-y)\Big(x^n(x-z) - y^n(y-z)\Big) + z^n(z-x)(z-y) \geq 0$$

Mathematical proofs **often** feature **wlog assumptions**

Theorem (Schur's inequality).

For all $x, y, z \geq 0$ and $n \in \mathbb{N}$, we have

$$x^n(x-y)(x-z) + y^n(y-x)(y-z) + z^n(z-x)(z-y) \geq 0$$

Proof.

Without loss of generality, assume $x \geq y \geq z$ and rearrange:

$$(x-y)\Big(x^n(x-z) - y^n(y-z)\Big) + z^n(z-x)(z-y) \geq 0$$

■

Mathematical proofs **often** feature **wlog assumptions**

Theorem (Schur's inequality).

For all $x, y, z \geq 0$ and $n \in \mathbb{N}$, we have

$$x^n(x-y)(x-z) + y^n(y-x)(y-z) + z^n(z-x)(z-y) \geq 0$$

Proof.

Without loss of generality, assume $x \geq y \geq z$ and rearrange:

$$(x-y)\Big(x^n(x-z) - y^n(y-z)\Big) + z^n(z-x)(z-y) \geq 0$$

■

Mathematical proofs **often** feature **wlog assumptions**

Theorem (Schur's inequality).

For all $x, y, z \geq 0$ and $n \in \mathbb{N}$, we have

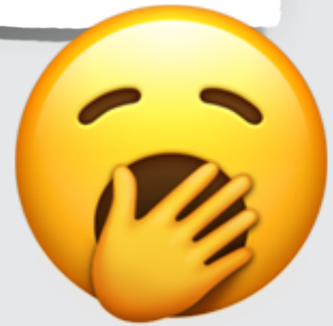
$$x^n(x-y)(x-z) + y^n(y-x)(y-z) + z^n(z-x)(z-y) \geq 0$$

Proof.

Without loss of generality, assume $x \geq y \geq z$ and rearrange:

$$(x-y)\Big(x^n(x-z) - y^n(y-z)\Big) + z^n(z-x)(z-y) \geq 0$$

■



Mathematical proofs **often** feature **wlog assumptions**

Theorem (Schur's inequality).

For all $x, y, z \geq 0$ and $n \in \mathbb{N}$, we have

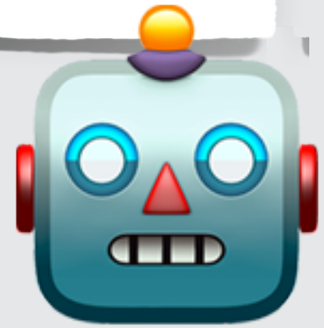
$$x^n(x-y)(x-z) + y^n(y-x)(y-z) + z^n(z-x)(z-y) \geq 0$$

Proof.

Without loss of generality, assume $x \geq y \geq z$ and rearrange:

$$(x-y)\Big(x^n(x-z) - y^n(y-z)\Big) + z^n(z-x)(z-y) \geq 0$$

■



Mathematical proofs **often** feature **wlog assumptions**

Theorem (Schur's inequality).

For all $x, y, z \geq 0$ and $n \in \mathbb{N}$, we have

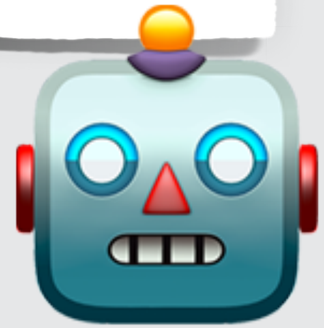
$$x^n(x-y)(x-z) + y^n(y-x)(y-z) + z^n(z-x)(z-y) \geq 0$$

Proof.

Without loss of generality, assume $x \geq y \geq z$ and rearrange:

$$(x-y)\Big(x^n(x-z) - y^n(y-z)\Big) + z^n(z-x)(z-y) \geq 0$$

■



Harrison (TPHOLs 2009) discussed **wlog** in the context of **HOL**

Without Loss of Generality

John Harrison

Intel Corporation, JF1-13
2111 NE 25th Avenue, Hillsboro OR 97124, USA
johnh@ichips.intel.com

Abstract. One sometimes reads in a mathematical proof that a certain assumption can be made 'without loss of generality' (WLOG). In other words, it is claimed that considering what first appears only a special case does nevertheless suffice to prove the general result. Typically the intuitive justification for this is that one can exploit symmetry in the problem. We examine how to formalize such 'WLOG' arguments in a mechanical theorem prover. Geometric reasoning is particularly rich in examples and we pay special attention to this area.

1 Introduction

Mathematical proofs sometimes state that a certain assumption can be made 'without loss of generality', often abbreviated to 'WLOG'. The phase suggest that although making the assumption at first sight only proves the theorem in a more restricted case, this does nevertheless justify the theorem in full generality. What is the intuitive justification for this sort of reasoning? Occasionally the phrase covers situations where we neglect special cases that are obviously trivial for other reasons. But more usually it suggests the exploitation of symmetry in the problem. For example, consider Schur's inequality, which asserts that for any nonnegative real numbers a, b and c and integer $k \geq 0$ one has $0 \leq a^k(a-b)(b-c) + b^k(b-a)(b-c) + c^k(c-a)(c-b)$. A typical proof might begin:

> Without loss of generality, let $a \leq b \leq c$.

If asked to spell this out in more detail, we might say something like:

> Since $\leq$ is a total order, the three numbers must be ordered somehow, i.e. we must have (at least) one of $a \leq b \leq c$, $a \leq c \leq b$, $b \leq a \leq c$, $b \leq c \leq a$, $c \leq a \leq b$ or $c \leq b \leq a$. But the theorem is completely symmetric between a, b and c, so each of these cases is just a version of the other with a change of variables, and we may as well just consider one of them.

Suppose that we are interested in formalizing mathematics in a mechanical theorem prover. Generally speaking, for an experienced formalizer it's rather routine to take an existing proof and construct a formal counterpart, even though it may require a great deal of work to get things just right and encourage the proof assistant check all the details. But with such 'without loss of generality' constructs, it's not immediately obvious what the formal counterpart should be. We can plausibly suggest two possible formalizations:

S. Berghofer et al. (Eds.): TPHOLs 2009, LNCS 5674, pp. 43–59, 2009.

Harrison (TPHOLs 2009) discussed **wlog** in the context of **HOL**

it would be more elegant to write a general parametrized proof script that we could use for all 6 cases with different parameters. This sort of programming is exactly the kind of thing that LCF-style systems [3] like HOL [2] are designed to make easy via their 'metalanguage' ML, and sometimes its convenience makes it irresistible. However, this approach is open to criticism on at least three grounds:

- Ugly/clumsy
- Inefficient
- Not faithful to the informal proof.

sometimes state that a certain assumption can be made 'without loss of generality', often abbreviated to 'WLOG'. The phase suggest that although making the assumption at first sight only proves the theorem in a more restricted case, this does nevertheless justify the theorem in full generality. What is the intuitive justification for this sort of reasoning? Occasionally the phrase covers situations where we neglect special cases that are obviously trivial for other reasons. But more usually it suggests the exploitation of symmetry in the problem. For example, consider Schur's inequality, which asserts that for any nonnegative real numbers a, b and c and integer $k \geq 0$ one has $0 \leq a^k(a-b)(b-c) + b^k(b-a)(b-c) + c^k(c-a)(c-b)$. A typical proof might begin:

> Without loss of generality, let $a \leq b \leq c$.

If asked to spell this out in more detail, we might say something like:

> Since $\leq$ is a total order, the three numbers must be ordered somehow, i.e. we must have (at least) one of $a \leq b \leq c$, $a \leq c \leq b$, $b \leq a \leq c$, $b \leq c \leq a$, $c \leq a \leq b$ or $c \leq b \leq a$. But the theorem is completely symmetric between a, b and c, so each of these cases is just a version of the other with a change of variables, and we may as well just consider one of them.

Suppose that we are interested in formalizing mathematics in a mechanical theorem prover. Generally speaking, for an experienced formalizer it's rather routine to take an existing proof and construct a formal counterpart, even though it may require a great deal of work to get things just right and encourage the proof assistant check all the details. But with such 'without loss of generality' constructs, it's not immediately obvious what the formal counterpart should be. We can plausibly suggest two possible formalizations:

S. Berghofer et al. (Eds.): TPHOLs 2009, LNCS 5674, pp. 43–59, 2009.

Harrison (TPHOLs 2009) discussed **wlog** in the context of **HOL**

it would be more elegant to write a general parametrized proof script that we could use for all 6 cases with different parameters. This sort of programming is exactly the kind of thing that LCF-style systems [3] like HOL [2] are designed to make easy via their 'metalanguage' ML, and sometimes its convenience makes it irresistible. However, this approach is open to criticism on at least three grounds:

- Ugly/clumsy
- Inefficient
- Not faithful to the informal proof.

sometimes state that a certain assumption can be made 'without loss of generality', often abbreviated to 'WLOG'. The phase suggest that although making the assumption at first sight only proves the theorem in a more restricted case, this does nevertheless justify the theorem in full generality. What is the intuitive justification for this sort of reasoning? Occasionally the phrase covers situations where we neglect special cases that are obviously trivial for other reasons. But more usually it suggests the exploitation of symmetry in the problem. For example, consider Schur's inequality, which asserts that for any nonnegative real numbers a, b and c and integer $k \geq 0$ one has $0 \leq a^k(a-b)(b-c) + b^k(b-a)(b-c) + c^k(c-a)(c-b)$. A typical proof might begin:

> Without loss of generality, let $a \leq b \leq c$.

If asked to spell this out in more detail, we might say something like:

> Since $\leq$ is a total order, the three numbers must be ordered somehow, i.e. we must have (at least) one of $a \leq b \leq c$, $a \leq c \leq b$, $b \leq a \leq c$, $b \leq c \leq a$, $c \leq a \leq b$ or $c \leq b \leq a$. But the theorem is completely symmetric between a, b and c, so each of these cases is just a version of the other with a change of variables, and we may as well just consider one of them.

Suppose that we are interested in formalizing mathematics in a mechanical theorem prover. Generally speaking, for an experienced formalizer it's rather routine to take an existing proof and construct a formal counterpart, even though it may require a great deal of work to get things just right and encourage the proof assistant check all the details. But with such 'without loss of generality' constructs, it's not immediately obvious what the formal counterpart should be. We can plausibly suggest two possible formalizations:

S. Berghofer et al. (Eds.): TPHOLs 2009, LNCS 5674, pp. 43–59, 2009.

Harrison (TPHOLs 2009) discussed **wlog** in the context of **HOL**

it would be more elegant to write a general parametrized proof script that we could use for all 6 cases with different parameters. This sort of programming is exactly the kind of thing that LCF-style systems [3] like HOL [2] are designed to make easy via their 'metalanguage' ML, and sometimes its convenience makes it irresistible. However, this approach is open to criticism on at least three grounds:

- Ugly/clumsy
- Inefficient
- Not faithful to the informal proof.

sometimes state that a certain assumption can be made 'without loss of generality', often abbreviated to 'WLOG'. The phase suggest that although making the assumption at first sight only proves the theorem in a more restricted case, this does nevertheless justify the theorem in full generality. What is the intuitive justification for this sort of reasoning? Occasionally the phrase covers situations where we neglect special cases that are obviously trivial for other reasons. But more usually it suggests the exploitation of symmetry in the problem. For example, consider Schur's inequality, which asserts that for any nonnegative real numbers a, b and c and integer $k \geq 0$ one has $0 \leq a^k(a-b)(b-c) + b^k(b-a)(b-c) + c^k(c-a)(c-b)$. A typical proof might begin:

> Without loss of generality, let $a \leq b \leq c$.

If asked to spell this out in more detail, we might say something like:

> Since $\leq$ is a total order, the three numbers must be ordered somehow, i.e. we must have (at least) one of $a \leq b \leq c$, $a \leq c \leq b$, $b \leq a \leq c$, $b \leq c \leq a$, $c \leq a \leq b$ or $c \leq b \leq a$. But the theorem is completely symmetric between a, b and c, so each of these cases is just a version of the other with a change of variables, and we may as well just consider one of them.

Suppose that we are interested in formalizing mathematics in a mechanical theorem prover. Generally speaking, for an experienced formalizer it's rather routine to take an existing proof and construct a formal counterpart, even though it may require a great deal of work to get things just right and encourage the proof assistant check all the details. But with such 'without loss of generality' constructs, it's not immediately obvious what the formal counterpart should be. We can plausibly suggest two possible formalizations:

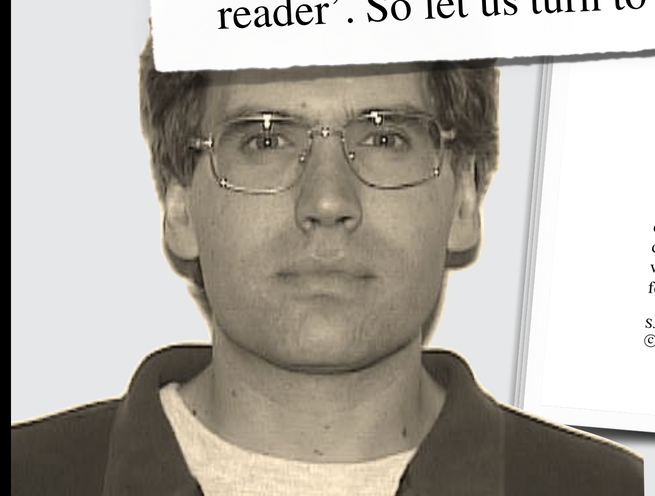
S. Berghofer et al. (Eds.): TPHOLs 2009, LNCS 5674, pp. 43–59, 2009.

Harrison (TPHOLs 2009) discussed **wlog** in the context of **HOL**

it would be more elegant to write a general parametrized proof script that we could use for all 6 cases with different parameters. This sort of programming is exactly the kind of thing that LCF-style systems [3] like HOL [2] are designed to make easy via their 'metalanguage' ML, and sometimes its convenience makes it irresistible. However, this approach is open to criticism on at least three grounds:

- Ugly/clumsy
- Inefficient
- Not faithful to the informal proof.

loss of generality' is meant to conjure up. If the book had intended that interpretation, it would probably have said something like 'the other cases are similar and are left to the reader'. So let us turn to how we might formalize and use a general logical principle.



Since $\leq$ is a total order, the three numbers must be ordered somehow, i.e. we must have (at least) one of $a \leq b \leq c$, $a \leq c \leq b$, $b \leq a \leq c$, $b \leq c \leq a$, $c \leq a \leq b$ or $c \leq b \leq a$. But the theorem is completely symmetric between a, b and c, so each of these cases is just a version of the other with a change of variables, and we may as well just consider one of them.

Suppose that we are interested in formalizing mathematics in a mechanical theorem prover. Generally speaking, for an experienced formalizer it's rather routine to take an existing proof and construct a formal counterpart, even though it may require a great deal of work to get things just right and encourage the proof assistant check all the details. But with such 'without loss of generality' constructs, it's not immediately obvious what the formal counterpart should be. We can plausibly suggest two possible formalizations:

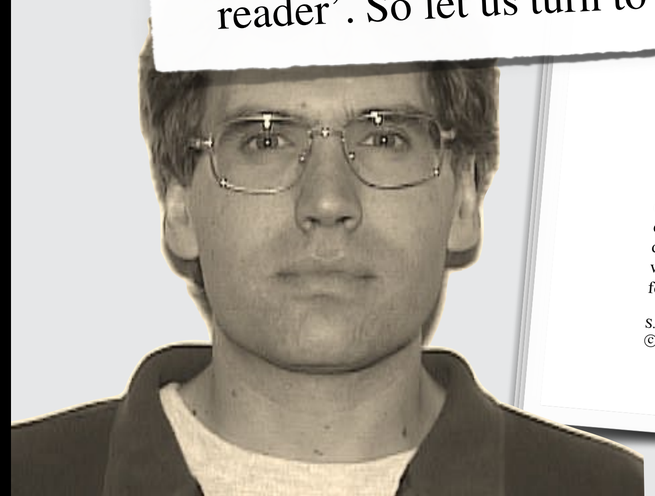
S. Berghofer et al. (Eds.): TPHOLs 2009, LNCS 5674, pp. 43–59, 2009.

Harrison (TPHOLs 2009) discussed **wlog** in the context of **HOL**

it would be more elegant to write a general parametrized proof script that we could use for all 6 cases with different parameters. This sort of programming is exactly the kind of thing that LCF-style systems [3] like HOL [2] are designed to make easy via their 'metalanguage' ML, and sometimes its convenience makes it irresistible. However, this approach is open to criticism on at least three grounds:

- Ugly/clumsy
- Inefficient
- Not faithful to the informal proof.

loss of generality' is meant to conjure up. If the book had intended that interpretation, it would probably have said something like 'the other cases are similar and are left to the reader'. So let us turn to how we might formalize and use a general logical principle.



Since $\leq$ is a total order, the three numbers must be ordered somehow, i.e. we must have (at least) one of $a \leq b \leq c$, $a \leq c \leq b$, $b \leq a \leq c$, $b \leq c \leq a$, $c \leq a \leq b$ or $c \leq b \leq a$. But the theorem is completely symmetric between a, b and c, so each of these cases is just a version of the other with a change of variables, and we may as well just consider one of them.

Suppose that we are interested in formalizing mathematics in a mechanical theorem prover. Generally speaking, for an experienced formalizer it's rather routine to take an existing proof and construct a formal counterpart, even though it may require a great deal of work to get things just right and encourage the proof assistant check all the details. But with such 'without loss of generality' constructs, it's not immediately obvious what the formal counterpart should be. We can plausibly suggest two possible formalizations:

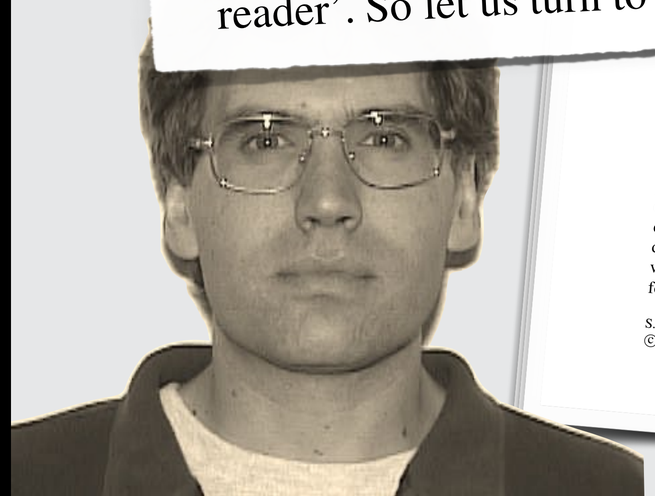
S. Berghofer et al. (Eds.): TPHOLs 2009, LNCS 5674, pp. 43–59, 2009.

Harrison (TPHOLs 2009) discussed **wlog** in the context of **HOL**

it would be more elegant to write a general parametrized proof script that we could use for all 6 cases with different parameters. This sort of programming is exactly the kind of thing that LCF-style systems [3] like HOL [2] are designed to make easy via their 'metalanguage' ML, and sometimes its convenience makes it irresistible. However, this approach is open to criticism on at least three grounds:

- Ugly/clumsy
- Inefficient
- Not faithful to the informal proof.

loss of generality' is meant to conjure up. If the book had intended that interpretation, it would probably have said something like 'the other cases are similar and are left to the reader'. So let us turn to how we might formalize and use a general logical principle.

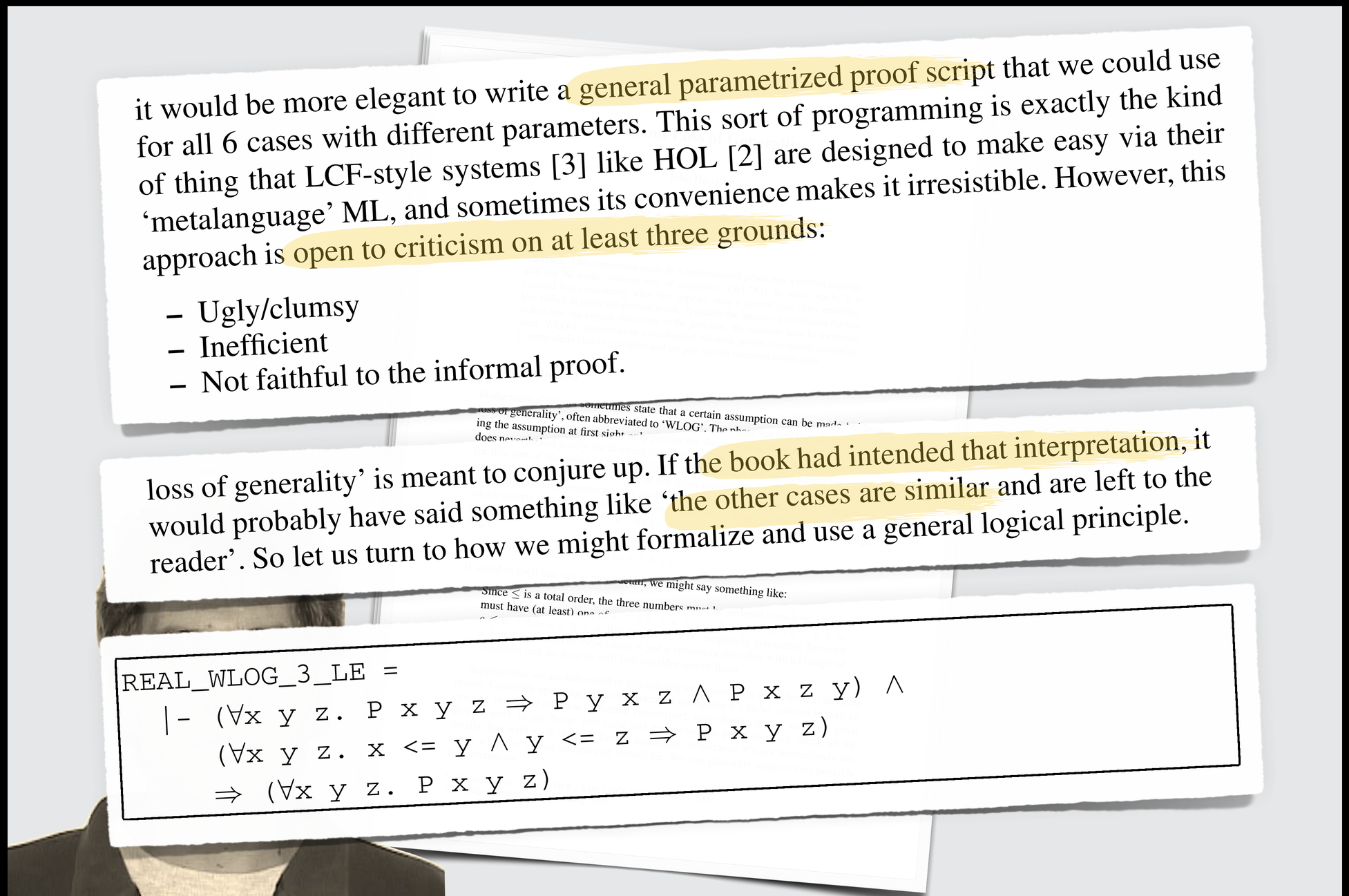


Since $\leq$ is a total order, the three numbers must be ordered somehow, i.e. we must have (at least) one of $a \leq b \leq c$, $a \leq c \leq b$, $b \leq a \leq c$, $b \leq c \leq a$, $c \leq a \leq b$ or $c \leq b \leq a$. But the theorem is completely symmetric between a, b and c, so each of these cases is just a version of the other with a change of variables, and we may as well just consider one of them.

Suppose that we are interested in formalizing mathematics in a mechanical theorem prover. Generally speaking, for an experienced formalizer it's rather routine to take an existing proof and construct a formal counterpart, even though it may require a great deal of work to get things just right and encourage the proof assistant check all the details. But with such 'without loss of generality' constructs, it's not immediately obvious what the formal counterpart should be. We can plausibly suggest two possible formalizations:

S. Berghofer et al. (Eds.): TPHOLs 2009, LNCS 5674, pp. 43–59, 2009.

Harrison (TPHOLs 2009) discussed **wlog** in the context of **HOL**



it would be more elegant to write a general parametrized proof script that we could use for all 6 cases with different parameters. This sort of programming is exactly the kind of thing that LCF-style systems [3] like HOL [2] are designed to make easy via their 'metalanguage' ML, and sometimes its convenience makes it irresistible. However, this approach is open to criticism on at least three grounds:

- Ugly/clumsy
- Inefficient
- Not faithful to the informal proof.

loss of generality' is meant to conjure up. If the book had intended that interpretation, it would probably have said something like 'the other cases are similar and are left to the reader'. So let us turn to how we might formalize and use a general logical principle.

```
REAL_WLOG_3_LE =
  |- (∀x y z. P x y z ⇒ P y x z ∧ P x z y) ∧
     (∀x y z. x <= y ∧ y <= z ⇒ P x y z)
     ⇒ (∀x y z. P x y z)
```

Or we can use a **simple** and **inefficient** method

$$x_1, x_2, x_3 \in \mathbb{R}$$

Or we can use a **simple** and **inefficient** method

$$x_1, x_2, x_3 \in \mathbb{R}$$

$\Downarrow$ sort with π

$$x_{\pi(1)} \leq x_{\pi(2)} \leq x_{\pi(3)}$$

Or we can use a **simple** and **inefficient** method

$$x_1, x_2, x_3 \in \mathbb{R}$$

$\Downarrow$ sort with π

$$x_{\pi(1)} \leq x_{\pi(2)} \leq x_{\pi(3)} \quad \xRightarrow{\text{actual proof}} \quad p : \texttt{Schur}(x_{\pi(1)}, x_{\pi(2)}, x_{\pi(3)})$$

Or we can use a **simple** and **inefficient** method

$x_1, x_2, x_3 \in \mathbb{R}$

$\Downarrow$ sort with π

$x_{\pi(1)} \leq x_{\pi(2)} \leq x_{\pi(3)}$

$\Longrightarrow$ actual proof

$p : \texttt{Schur}(x_{\pi(1)}, x_{\pi(2)}, x_{\pi(3)})$

$\Uparrow$ revert the sort as $\texttt{Schur}$ is invariant

$p' : \texttt{Schur}(x_1, x_2, x_3)$

IS THERE A
BIGGER picture?

YES!

GROUPS!

YES!

GROUPS!

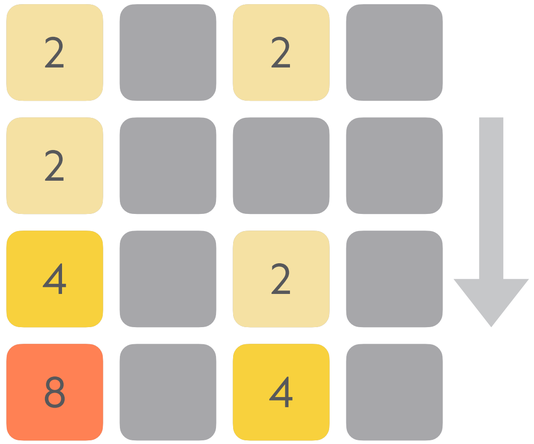
Symmetries form a **group**

Group G

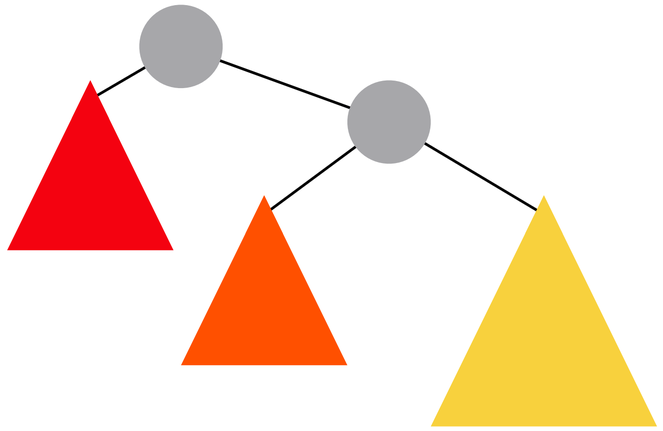
- a *carrier* set $\underline{G}$
- *unit* element $e : \underline{G}$
- *multiplication* $* : \underline{G} \to \underline{G} \to \underline{G}$
- *inverse* $(\,\cdot\,)^{-1} : \underline{G} \to \underline{G}$

such that:

- $e * g = g * e = g$
- $(g * h) * k = g * (h * k)$
- $g * g^{-1} = e$



$\mathrm{Dih}_4 = \langle \sigma_x, \rho_{90^\circ} \mid \cdots \rangle$



$\underline{\mathrm{Sym}_2} = \{I, F\}$

Schur's inequality

$\underline{\mathrm{Sym}_3} = \{(123), (132), (213), (231), (312), (321)\}$

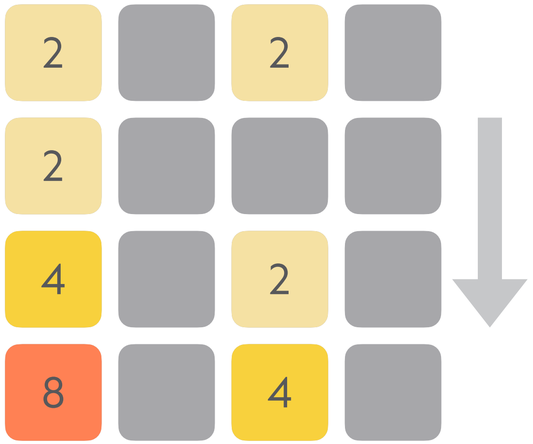
Sets **possesing a symmetry** are captured with **actions**

G-action A

- a *carrier* set $\underline{A}$
- a map $\circledast : \underline{G} \to \underline{A} \to \underline{A}$

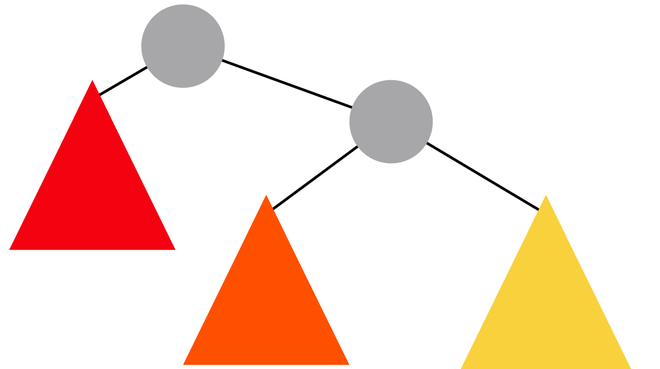
such that:

- $e \circledast x = x$
- $(g * h) \circledast x = g \circledast (h \circledast x)$



$$\sigma_x \circledast [t_{i,j}]_{ij} = [t_{4-i,j}]_{ij}$$

$$\rho_{90^\circ} \circledast [t_{i,j}]_{ij} = [t_{4-j,i}]_{ij}$$



$$I \circledast t = t$$

$$F \circledast L = L$$

$$F \circledast N(t_1, x, t_2) = N(F \circledast t_2, x, F \circledast t_1)$$

For dependent sets, we employ **indexed actions**

A-action B

- a *carrier* family $\underline{B} : \underline{A} \to \mathbf{Set}$
- a map

$$\circledast : (g : \underline{G}) \to (x : \underline{A}) \to \underline{B}x \to \underline{B}(g \circledast x)$$

such that:

- $e \circledast_x y = y$
- $(g * h) \circledast_x y = g \circledast_{h \circledast x} (h \circledast_x y)$

Schur's inequality

$$\underline{A} = \mathbb{R}^3$$

$$\pi \circledast (x_1, x_2, x_3) = (x_{\pi(1)}, x_{\pi(2)}, x_{\pi(3)})$$

$$\underline{B}(x_1, x_2, x_3) = \mathtt{Schur}(x_1, x_2, x_3)$$

$$\pi \circledast \big(p : \mathtt{Schur}(x_1, x_2, x_3)\big) = \big(q : \mathtt{Schur}(x_{\pi(1)}, x_{\pi(2)}, x_{\pi(3)})\big)$$

For dependent sets, we employ **indexed actions**

A-action B

- a *carrier* family $\underline{B} : \underline{A} \to \mathbf{Set}$
- a map

$$\circledast : (g : \underline{G}) \to (x : \underline{A}) \to \underline{B}x \to \underline{B}(g \circledast x)$$

such that:

- $e \circledast_x y = y$
- $(g * h) \circledast_x y = g \circledast_{h \circledast x} (h \circledast_x y)$

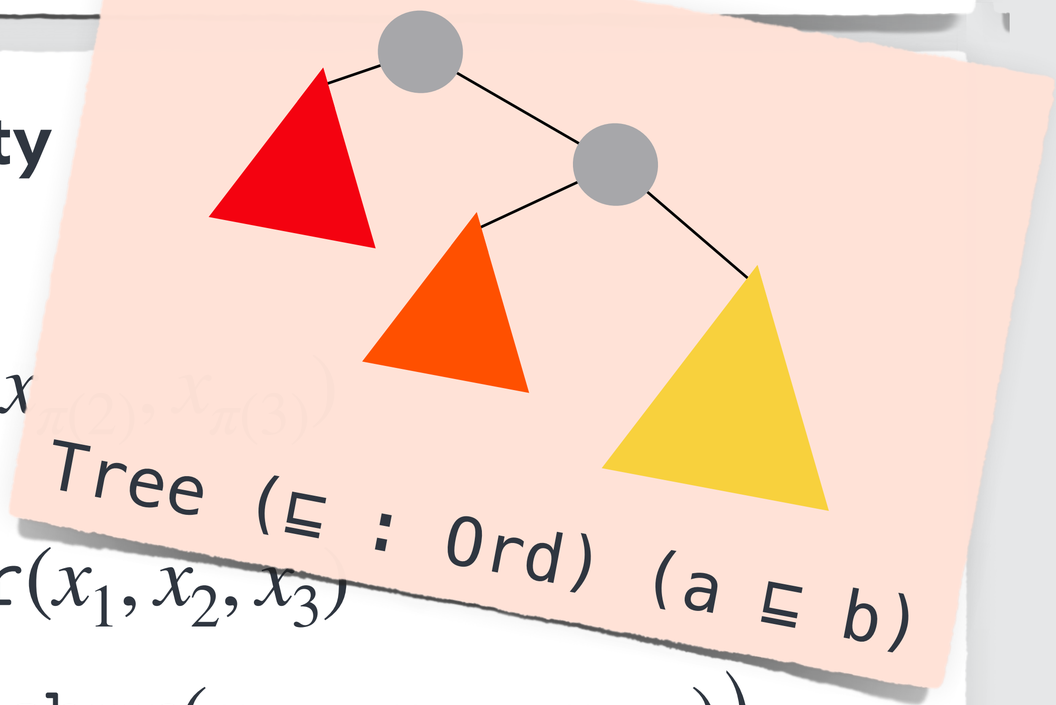
Schur's inequality

$$\underline{A} = \mathbb{R}^3$$

$$\pi \circledast (x_1, x_2, x_3) = (x_{\pi(1)}, x_{\pi(2)}, x_{\pi(3)})$$

$$\underline{B}(x_1, x_2, x_3) = \mathtt{Schur}(x_1, x_2, x_3)$$

$$\pi \circledast \big(p : \mathtt{Schur}(x_1, x_2, x_3)\big) = \big(q : \mathtt{Schur}(x_{\pi(1)}, x_{\pi(2)}, x_{\pi(3)})\big)$$

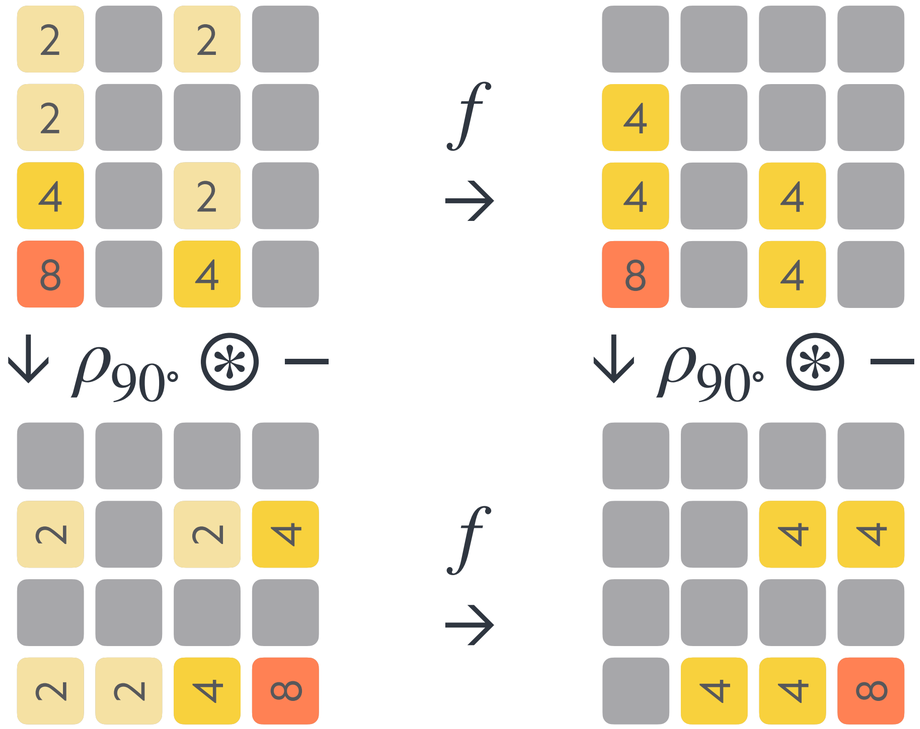


Maps between symmetric sets are **equivariant**

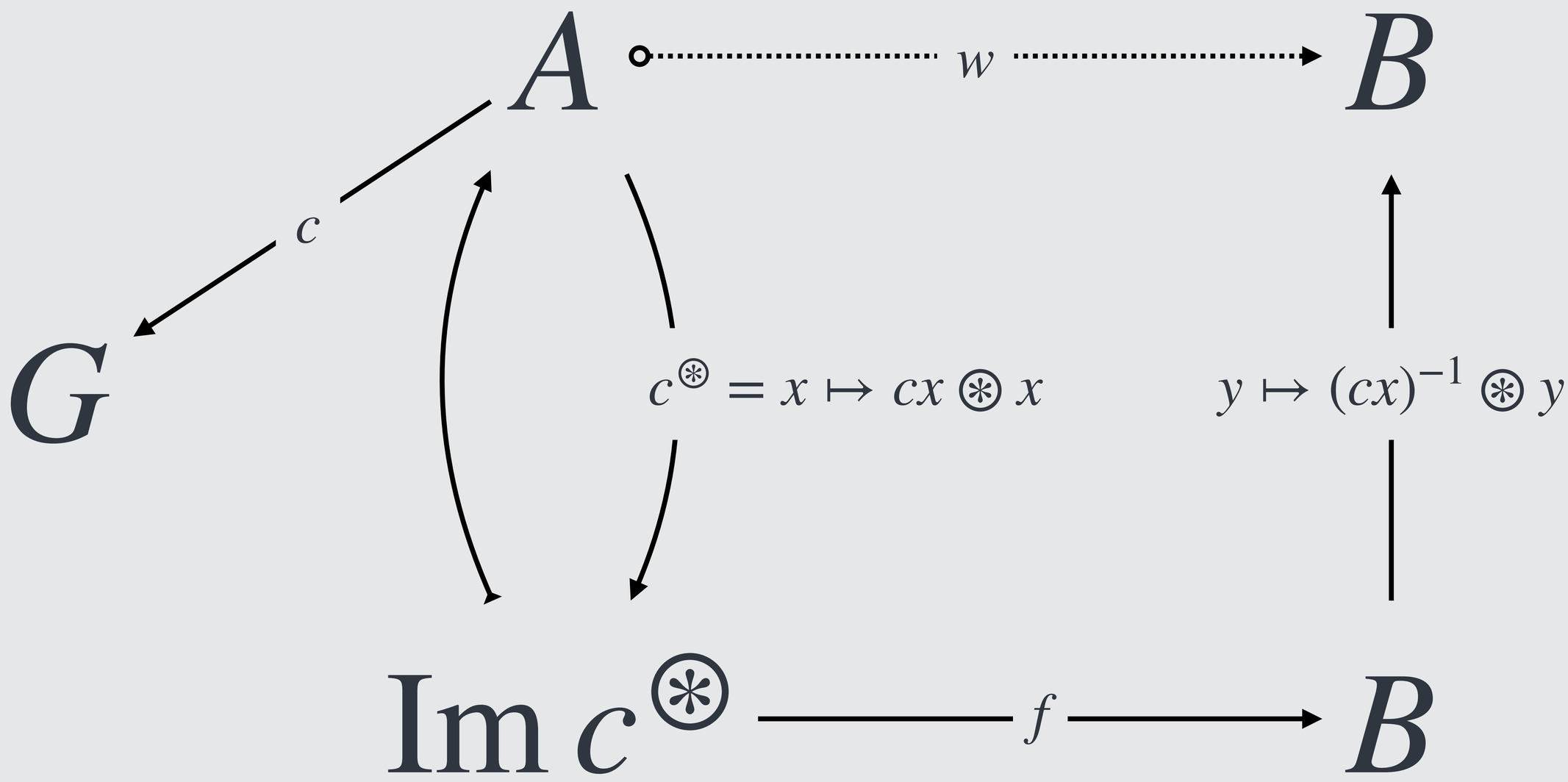
equivariant map $f : A \circledast\!\!\rightarrow B$

$$f : (x : \underline{A}) \to \underline{B}x$$

$$f(g \circledast x) = g \circledast_x f(x)$$



Construction of **w.l.o.g**



Construction of **w.l.o.g**

Theorem

Take:

- *a group* G*, a* G*-action* A*, and an* A*-indexed action* B
- *a* canoniser assignment map $c : \underline{A} \to \underline{G}$
- *a map* $f : (x : \operatorname{Im} c^{\circledast}) \to Bx$ *such that*

$$\forall g \in \underline{G}, x \in \operatorname{Im} c^{\circledast} \,.\, (g \circledast x \in \operatorname{Im} c^{\circledast}) \implies f(g \circledast x) = g \circledast_{x} f(x)$$

Then, there exists a unique equivariant map $w : A \circledast\!\!\to B$ *extending* f*, defined as:*

$$w(x) := (cx)^{-1} \circledast_{cx \circledast x} f(cx \circledast x)$$

Can we **reduce** the set of canonical elements?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

Can we **reduce** the set of canonical elements?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

$$\operatorname{Fix} c^{\circledast} = \{x \in A \mid cx \circledast x = x\}$$

Can we **reduce** the set of canonical elements?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

$$\operatorname{Fix} c^{\circledast} = \{x \in A \mid cx \circledast x = x\}$$

$$\operatorname{Ker} c = \{x \in A \mid cx = e\}$$

Can we **reduce** the set of canonical elements?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

$$\cup\!| \quad |\!\cup$$

$$\operatorname{Fix} c^{\circledast} = \{x \in A \mid cx \circledast x = x\}$$

$$\operatorname{Ker} c = \{x \in A \mid cx = e\}$$

Can we **reduce** the set of canonical elements?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

$$\cup\!|$$

$$\operatorname{Fix} c^{\circledast} = \{x \in A \mid cx \circledast x = x\}$$

$$\cup\!|$$

$$\operatorname{Ker} c = \{x \in A \mid cx = e\}$$

When do the **converses** hold?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

$$\cap\!|$$

$$\operatorname{Fix} c^{\circledast} = \{x \in A \mid cx \circledast x = x\}$$

$$\cap\!|$$

$$\operatorname{Ker} c = \{x \in A \mid cx = e\}$$

When do the **converses** hold?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

$\cap\!\!|$ $\Leftrightarrow$ $c^{\circledast}$ is idempotent

$$\operatorname{Fix} c^{\circledast} = \{x \in A \mid cx \circledast x = x\}$$

$\cap\!\!|$

$$\operatorname{Ker} c = \{x \in A \mid cx = e\}$$

When do the **converses** hold?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

$\bigcap\!\!\mid \Leftrightarrow c^{\circledast}$ is idempotent

$$\operatorname{Fix} c^{\circledast} = \{x \in A \mid cx \circledast x = x\}$$

$\bigcap\!\!\mid \Leftarrow A$ is semi-regular

$$\operatorname{Ker} c = \{x \in A \mid cx = e\}$$

When do the **converses** hold?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

$$\cap\!|$$

$$\operatorname{Ker} c = \{x \in A \mid cx = e\}$$

When do the **converses** hold?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

$$\bigcap\!| \Leftrightarrow \forall x, c(cx \circledast x) = e$$

$$\operatorname{Ker} c = \{x \in A \mid cx = e\}$$

When do the **converses** hold?

$$\operatorname{Im} c^{\circledast} = \{cx \circledast x \mid x \in A\}$$

$$\bigcap\!| \Leftrightarrow \forall x, c(cx \circledast x) = e$$

$$\operatorname{Ker} c = \{x \in A \mid cx = e\}$$

Our principle implies **Harrison's tactics**

```
REAL_WLOG_3_LE =
  |- (∀x y z. P x y z ⇒ P y x z ∧ P x z y) ∧
     (∀x y z. x <= y ∧ y <= z ⇒ P x y z)
     ⇒ (∀x y z. P x y z)
```

Our principle implies **Harrison's tactics**

P has an Sym_3-indexed action

```
REAL_WLOG_3_LE =
  |- (∀x y z. P x y z ⇒ P y x z ∧ P x z y) ∧
     (∀x y z. x <= y ∧ y <= z ⇒ P x y z)
     ⇒ (∀x y z. P x y z)
```

Our principle implies **Harrison's tactics**

P has an Sym_3-indexed action

```
REAL_WLOG_3_LE =
   |- (∀x y z. P x y z ⇒ P y x z ∧ P x z y) ∧
      (∀x y z. x <= y ∧ y <= z ⇒ P x y z)
      ⇒ (∀x y z. P x y z)
```

$f : (x, y, z) : \mathrm{Ker}\, c \to P(x, y, z)$

What's **next?**

What's **next?**

soon

- more examples
- implementation

What's **next?**

soon

- more examples
- implementation

later

- efficiency
- programming language
- symmetric data types